TAL TECH

DOCTORAL THESIS

Consolidation of Crowd-Sourced Geo-Tagged Data for Parameterized Travel Recommendations

Ago Luberg

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY
TALLINN 2021

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
23/2021

Consolidation of Crowd-Sourced Geo-Tagged Data for Parameterized Travel Recommendations

AGO LUBERG

TAL TECH PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

The dissertation was accepted for the defence of the degree of Doctor of Philosophy in Computer Science on April 19th, 2021

Supervisor: PhD Tanel Tammet,
Department of Software Science, School of Information Technologies,
Tallinn University of Technology
Tallinn, Estonia

Opponents: Dr. Chiara Renso,
Senior Researcher at ISTI-CNR,
Pisa, Tuscany, Italy

Dr. Wolfgang Wörndl,
Technical University of Munich,
München, Germany

Defence of the thesis: May 27th 2021, Tallinn

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted for any academic degree elsewhere.

Ago Luberg _____
signature

ISSN 2585-6898 (publication)
ISBN 978-9949-83-691-8 (publication)
ISSN 2585-6901 (PDF)
ISBN 978-9949-83-692-5 (PDF)
Printed by Koopia Niini & Rauam

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
23/2021

Kasutajatelt kogutud andmete integreerimine parametriseeritud reisisoovituste loomiseks

AGO LUBERG

TAL TECH
KIRJASTUS

Contents

List of Publications

The present Ph.D. thesis is based on the following publications that are referred to in the text by Roman numbers.

I A. Luberg, J. Pindis, and T. Tammet. Sights, titles and tags: Mining a worldwide photo database for sightseeing. In *Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics*, WIMS 2020, page 149–158, New York, NY, USA, 2020. Association for Computing Machinery

II T. Tammet, A. Luberg, and P. Järv. Sightsmap: Crowd-sourced popularity of the world places. In L. Cantoni and Z. P. Xiang, editors, *Information and Communication Technologies in Tourism 2013*, pages 314–325, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg

III A. Luberg, M. Granitzer, H. Wu, P. Järv, and T. Tammet. Information retrieval and deduplication for tourism recommender sightsplanner. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, WIMS '12, New York, NY, USA, 2012. Association for Computing Machinery

IV A. Luberg, P. Järv, and T. Tammet. Information extraction for a tourist recommender system. In M. Fuchs, F. Ricci, and L. Cantoni, editors, *Information and Communication Technologies in Tourism 2012*, pages 332–343, Vienna, 2012. Springer Vienna

V A. Luberg, P. Järv, K. Schoefegger, and T. Tammet. Context-aware and multilingual information extraction for a tourist recommender system. In *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies*, i-KNOW '11, New York, NY, USA, 2011. Association for Computing Machinery

VI A. Luberg, T. Tammet, and P. Jarv. Extended triple store structure used in recommender system. In *2011 22nd International Workshop on Database and Expert Systems Applications*, pages 539–543, Aug 2011

VII A. Luberg, T. Tammet, and P. Järv. Smart city: A rule-based tourist recommendation system. In R. Law, M. Fuchs, and F. Ricci, editors, *Information and Communication Technologies in Tourism 2011*, pages 51–62, Vienna, 2011. Springer Vienna

Author's Contributions to the Publications

I In I, I was the main author, worked out the methodology for tagging, and wrote the manuscript.

II In II, I was responsible for labelling and merging section, also carried out the experiments.

III In III, I was the main author, gathered the data, conducted the experiments, analysed the results, and wrote the manuscript

IV In IV, I was the main author, worked out the methodology for clustering and keyword extraction, conducted the experiments, and wrote the manuscript.

V In V, I was the main author, worked out the methodology for scraping, keyword extraction and data merging, and wrote the manuscript.

VI In VI, I was the main author, worked out the methodology for storing data in extended triple storage structure and score calculation between user the profile and the tourism object, and wrote the manuscript.

VII In VII, I was the main author, worked out the methodology for clustering and keyword extraction, conducted the experiments, and wrote the manuscript.

Other Publications

Other publications that the author has contributed to during the studies at the Tallinn University of Technology.

VIII A. Kuusik, E. Reilent, I. Lõõbas, and A. Luberg. Data acquisition software architecture for patient monitoring devices. *Elektronika ir Elektrotechnika*, 105(9):97–100, Jan. 2010

IX H. Wu, A. Luberg, and T. Tammet. Ranking domain objects by wisdom of web pages. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, pages 1–4, 2012

X T. Tammet and A. Luberg. Combining fuzzy and probabilistic reasoning for crowd-sourced categorization and tagging. In *Web Reasoning and Rule Systems: 8th International Conference, RR 2014, Athens, Greece, September 15-17, 2014. Proceedings*, volume 8741, page 247. Springer, 2014

XI I. Lõõbas, E. Reilent, A. Anier, A. Luberg, and A. Kuusik. Towards semantic contextual content-centric assisted living solution. In *The 12th IEEE International Conference on e-Health Networking, Applications and Services*, pages 56–60. IEEE, 2010

Abbreviations

NLP	Natural Language Processing
OWL	Web Ontology Language
POI	Place of Interest (or Point of Interest)
RDF	Resource Description Framework
SVM	Support-Vector Machine
URI	Uniform Resource Identifier

Terms

Deduplication	The process of removing duplicate objects.
Heatmap	Map visualisation where more popular areas are highlighted (represented with brighter color).
Merging	The process of combining duplicate objects into one.

List of Figures

1 Introduction

The general subject area of this dissertation is automated creation of recommendations for tourism, with the focus on sights and visit-worthy locations. This stands in contrast to the recommendations provided by mainstream travel systems, which focus on commercial services like hotels, restaurants and travel tickets. Creating tourism recommendations is a widely studied and a heavily commercialised area: there are numerous publications considering different aspects of creating such recommendations and a large number of widely used software systems available.

Nevertheless, from the practical standpoint the problem of creating recommendations for sights is largely unsolved: from personal and widely recognized anecdotal experience we claim that in most cases the tourists are not following the recommendations of automated systems, for various reasons. Regardless of the concrete reasons for each particular tourist we also claim - from similar grounds - that the majority of automated recommendation systems are not intelligent enough and do not have enough information about the potential locations and actual preferences and limitations of concrete tourists, thus leading to a general distrust of such systems on several levels.

This distrust often manifests itself in little interest in actually deploying fully automated systems on widely used tourism sites: instead, the large public tourism systems mostly rely on human-curated lists of top sights, often split into several main categories. The reliance on human-curated lists stems from various reasons: the perceived inadequacy of fully automated systems, the wish to augment and tune the descriptions of sites according to the perceived users of tourism sites, the commercial and organizational interests of the maintainers of the tourism sites.

Despite the overwhelming reliance on human-curated recommendations, these recommendations are often missing important sites or show heavy bias towards certain kinds of sites, mainly the ones which provide paid services, in contrast to visually interesting sites without such services.

The goal of the research presented in the thesis is to investigate ways to improve several critical weaknesses of fully automated systems, thus laying the groundwork for developing better automated systems. Our approach is a combination of engineering-oriented research demonstrating the feasibility of certain methods and algorithms and empirical research showing the measurable improvements gained by the algorithms we have developed. Conducting social experiments for demonstrating the actual usefulness of this or that method to real tourists is outside the scope of our research.

It is also important to note that there are two main approaches to recommendation systems: the collaborative filtering approach and the content-based approach. The collaborative filtering approach is based on finding other users which have stated similar explicit preferences to a current user. The other preferences of these similar users are then suggested as likely preferences for the current user. This approach works well in case a large database of actual user preferences is available along with the record of the preferences of the current user, as is common in various web-based sales systems. In case when such a database and/or previous preferences are not available - the so called "cold start problem" - collaborative filtering is impossible and we have to base our recommendations on analysing the generic contents and/or popularity of the recommended object. This content-based approach is the main context of our research.

The first hard question for the tourism recommenders we have worked on appears to be collecting comprehensive and high-quality information about interesting places to visit, including information like popularity, different categories of places and their weights (categories of places tend to be fuzzy), human-readable content, suitable names, etc. Our

experience in different projects indicates that the most promising way to collect quality data is to harvest and consolidate different databases, including curated information, wikipedia content, wikipedia usage statistics, large crowd-sourced photo databases like Panoramio, Flickr, Instagram, check-in statistics for crowd-sourced places like Foursquare and Swarm etc. Notice that the specifics of these crowd-sourced systems provide us with different aspects of places. For example, the density of photos taken in some area indicates its visual attractiveness, the log count of visits to corresponding Wikipedia places indicates how well the places are known, the checkins in systems like Foursquare/Swarm indicate the actual number of visits.

Thus the main focus on our research became consolidation of information from various sources: names, locations, descriptions, photo titles, different types of popularities, etc. One question with particular interest to us is the problem of detecting duplicate objects in databases: the process which is sometimes called deduplication. We develop a learning method for optimizing the parameters of a deduplication algorithm and show that it leads to significant improvement over hand-crafted methods.

The current thesis is structured as a collection of publications, hence the introduction is kept short. The reader is advised to look into the included research papers for the actual goals, details and results of the research.

1.1 Problem statement and research contribution

Place recommendation in tourism recommender systems relies on a model built from a number of varied sources. Before any recommendations can be calculated, a recommender system needs to know interesting points of interest (POIs) along with the most suitable name and membership in POI categories. This is a surprisingly hard problem not solved well enough by existing databases and recommender systems.

Why are these questions hard? First, no authoritative and universally suitable databases of touristically interesting POIs exist. Data has to be gleaned and integrated from widely different databases, which may, for example, provide a number of widely different names, categories and other specifics for a seemingly single POI or an area. An important specific question is determining whether POIs obtained from different sources are actually the same object or two different objects. Second, POIs range from very small (a single statue) to very large (a whole city), which makes both the comparison of POIs and feature selection complicated: different POIs with different names and categories should be shown for different zoom levels of a map.

The main contributions of the thesis focus on developing algorithms and machine learning methods for automatic disambiguation of points of interest (POIs) and finding the most suitable names and categories of POIs for tourism recommender systems, using both very large databases of crowd-sourced photo titles (Panoramio and Flickr) and semi-structured crowd-sourced descriptions of known POIs (Wikipedia, Foursquare, GeoNames) for input data:

- Developed a frequency based methodology and algorithms to find tags/categories for tourist places on the example of Panoramio picture titles. The developed algorithm does not require any pre-trained model in order to find suitable tags and category candidates. The methodology and algorithm is described in I.
- Comparison of the accuracy of different tag recommendation methods referred to in the literature. Demonstrating that the algorithm developed is a top performer. Our algorithm does not require external knowledge base which makes it fast and convenient to use. The comparison is presented in I.

- Developed a taxonomy of POI categories suitable for worldwide tourism use, based on the frequency of tags in the whole set of Panoramio photos. Using pictures all around the world, we were able to collect a convenient set of categories. We have organised found categories and developed the taxonomy which is used to categorise objects in Sightsmap application. The developed taxonomy is presented in I.

- Developed an algorithm for finding most suitable names for POIs from the large set of photo titles. The names were used by a tourism recommender system. The same Panoramio dataset was used as with the categories. The work about finding names is presented in II.

- Determining and comparing the importance of different POI features for disambiguating non-frequent geo-objects scraped from different web portals. Different features were measured separately and in combination with other features. The overview can be seen in III.

- Evaluation of the training set selection heuristics, clustering and optimization of distance functions for machine learning of POI disambiguation. Using machine learning improved our manual deduplication accuracy from 85% to 98%. The process is described in more detail in III.

- Developed a rule-based calculation algorithm with confidence scores for evaluating the suitability of POIs for personal topic-based recommendation and membership of POIs in fuzzy categories. We have shown how to represent user interests with confidence scores which can be matched against tourism object categories. The calculation mechanism is presented in VII, the algorithm is presented in VI.

The results of this work have been directly applied in three different recommender systems: http://sightsmap.com, which is currently unmaintained due to the removal of the Panoramio photo database from the public web by Google, http://sightsplanner.com, which is not available any more, and http://visitestonia.com, which used a topic-based recommender system for some time.

1.2 Structure of the dissertation

Chapter 3 gives an overview of different projects which has motivated the given thesis. This chapter describes how the projects are evolved into more general system and how those are related to presented publications. The chapter also gives a broader sense how the different parts are combined into a larger system of tourism related tools.

The thesis covers different topics in the process of providing recommendations for tourists. The rest of the document is divided into chapters for all the covered topics. Each of the chapters provides an introduction and describes the work covered in the related publications. The related work is presented in the chapter 2 which is divided into subsections by the different topics.

Chapter 4 gives an overview of data acquisition and how the information is extracted. Different data sources are described along with the process of gathering data semi-automatically from different sources.

Chapter 5 presents the work on data deduplication. Gathering data from different sources yields in duplicate information about the same objects. The chapter explores the problems and presents different stages of deduplication: manually configured and another which improves the results of manual setup with machine learning.

Chapter 6 introduces automatic tagging and naming of the objects based on the image titles. The chapter describes the process of clustering geo-tagged images by their location and then finding categories and names for those clusters. Combined clusters are merged together with objects from other data sources. The chapter also compares different tag recommendation methods with our proposed one.

Chapter 7 presents how the data is used to make recommendations for tourists. The chapter introduces object score calculation based on the user profile and the object category information. The users specify their interests about different topics or categories. The interest is measured as a value in the range $[0, 1]$. The same range is used to indicate the confidence of the object having the given property. Those scores are then used to calculate a total score for a tourist for an object which indicates the interestingness of the given object for the given tourist. The chapter describes how this interestingness score is used to recommend the itinerary based on the location and scores.

2 Related work

2.1 Data retrieval

A lot of work has been done in describing the benefits of Semantic Web and how it could be used to improve the quality of information retrieval. Kiryakov et al. in [14] describe how specific knowledge is required in order to make sense of data and its structure. They discuss the possibility to use language processing to structure data and find named entities.

In [29] they describe a recommender system which regularly updates information about events from regular non-structured web pages. For each source there is a separate configuration, and a script which identifies different segments from HTML page and downloads the contents. They apply different NLP methods (named entity recognition, shallow parsing etc.) to the texts to find the title, location, price and the time of the event. After processing they fill the required fields, for example resolve address into geographical location.

Kara et al.[11] describe ontology-based information retrieval system in the soccer domain. They use web portals to gather textual information about soccer matches. They have designed their own soccer ontology which describes all the required concepts of the game play. The ontology is used in information extraction and also inference steps. They make use of a reasoner to deduct new facts about the game play. For example, they can infer an assist for the goal taken into account the pass to the goal scorer in the same minute as the goal was shot. In [49] they show how they extract named entities from Turkish text. They use manually formed patterns to extract information from unstructured documents. They point out that their approach can be applied to any language, as they are not using POS (part-of-speech) tagging, phrase chunker etc.

Salas-Olmendo et al.[41] use data from Panoramio, Foursquare and Twitter to compare tourist activity in one city (Madrid, Spain). Each data source has crowd-source content provided with the geo-location. The location is used to indicate popular places inside the city. Based on the data source, they differentiate between popular areas of sightseeing (Panoramio), tourist consumption (Foursquare) and Internet consumption (Twitter). They have also used the data to draw density maps for each data source.

Soualah-Alila et al.[43] describe how it is important to have a standard for data representation in order to achieve interoperability. They propose their own ontology TIFSem to describe tourism objects. The proposed ontology [44] is mapped to Schema.org [1] which makes it more close to the standard. They process the text using Gramlab Unitex [34] corpus processor which analyses natural language texts. After the text is cleaned, events are annotated using TimeML language [36] to indicate temporal information along with relational propositions like "before", "after" etc.

Patroumpas et al.[33] propose an OWL-ontology to represent POI information in RDF. They use a tool TripleGeo [32] which can turn geospatial data into RDF. The tool has been updated to support the proposed ontology. TripleGeo can be used to transform different types of sources into the ontology, but also supports reverse transformation from ontology to conventional POI formats. The tool supports a wide variety of input types from structured and geospatially-enabled database systems. For reverse transformation they admit that their proposed ontology is semantically more expressive than conventional POI schemata. Therefore, reverse transformation currently does not support the full range of the proposed ontology.

2.2 Deduplication

In [29] they scrape web pages regularly in order to get events for recommender system. One step of the process is deduplication. They apply Vector Space Model for text content and use the cosine similarity to calculate similarity between two events. If the calculated similarity is above certain threshold, the new event will be discarded. Another threshold is set to indicate that the event has to be validated by an administrator.

In [52] authors detect duplicate records in geographical objects from yellow pages. They have introduced 4 similarity functions or features: two title comparisons, address similarity and category similarity. Authors apply machine learning to find duplicate pairs. In the paper they have compared each feature separately and also compared different combinations of the features.

Kim et al.[13] describe the methodology to use the description of a place to find its relative location compared to surrounding other places. Parsing the textual data using natural language processing they construct graphs which describe the relation of objects. The graphs are used to indicate similarity of objects. In the case where two objects have similar names and their neighbours are similar, then the places are considered as duplicates and could be combined into one.

Koumarelas et al.[15] are dealing with deduplication of addresses. They apply geocoding and other services to enrich the data. For example, from the given address string, they apply geocoding to get the corresponding latitude and longitude. Then they use reverse geocoding to get the address information which already has more detailed information. After enriching the addresses, they calculate similarity between two addresses.

2.3 Tagging

The tagging / naming method we propose and employ uses a large number of photos to automatically tag popular touristic places. We do not use the actual visual contents of the pictures: instead we use the user-defined titles along with the location. A significant amount of work has been done in the area of image recognition in order to detect the objects in the pictures. A similarly popular area of research is to tag text (news, research papers etc.). Our case is more specific: we do not use picture contents and we do not have long texts to work with. We will give an overview of related work which covers ideas similar to ours.

Authors in [7] have described photo annotation based on the different goals. One goal is to detect "where?" the picture has been taken. They describe it as location of the photo. The annotation can have different senses. The most common sense is the definition of the concrete object or geographical position. For example a postal address or the name of the object/place ("Eiffel Tower", "Paris"). Another sense can relate to a concrete person ("My home", "My work") - those mean different things for different users. Also, a spatial reference of a point of interest is often used ("near Eiffel Tower"). As pointed out, all those describe the location in a different way. With the personal reference ("My home"), there is no way to determine the actual object in the picture. Also, we do not use address information in our case. We are mainly interested in the descriptive words in the title which point out the type of the object in the picture.

Another point which is made by de Andrade et al. is that often the geocoordinates of the picture (the point where the camera is) does not match with the location of the object itself. For example, it is common to take pictures of Eiffel Tower few hundreds of meters away in order to fit the whole tower along with the surroundings. If we annotate the point where the pictures are taken as "Eiffel Tower" it does not indicate the actual object itself,

but rather a place where you can enjoy (take pictures) the view of the object. As in our case, we use a visual representation of different objects, we can say that each location is annotated with the tags which are visible at that point.

Authors in [45] use images from Flickr social network to indicate spatial areas of interest and events. They divide an area of predefined bounding box (a city center for example) into smaller groups of fixed size called geo clusters. Flickr photos have user defined tags which they use to identify the places of interests. By analyzing the textual data, they can rank the tags and select the most popular to describe the places and events. They analyze the time when the photos are taken. If a lot of images are within a short range of time, those probably describe an event. Landmarks generally are of interest the whole year, so they expect that photos should distribute uniformly through time.

Spyrou et al. define a model to describe the rank of the tag. They divide images into clusters and they use the images inside the cluster to find the best tags to describe the whole cluster. Tags which appear in more images in the cluster rank higher. Tags which appear in in several clusters have lower importance than those which are unique to a certain cluster. In addition to a previously mentioned measures, they take into account the authors of the images. Tags which appear with the images taken by a large number of users rank higher than those which are ralted to small number of users. They argue that a typical user could upload a lot of images of the same object and use the same title/tag for those. If the number of the uploaded images is high enough, it can bias the tag to have a higher ranking. They also try to use the spatial neighbors of images to tag untagged images. In the end they combine all the mentioned rankings into one ranking measure for a tag inside a cluster.

When dealing with a set of tags, they combine the tags which are similar. They use Levenshtein distance [19] and cosine similarity to group similar tags. If the similarity is above a predefined threshold, tags are grouped into one and the most frequent one is a representative of this group which will be used in ranking.

A very detailed and thorough automatic picture analysis is presented in [37]. They have described the whole pipeline on how to cluster geotagged pictures, classify those and how to link clusters with Wikipedia. They use a subset of European cities and extract about 220,000 pictures for those places. For clustering they use both the visual and text-based similarity. They point out that if they used only text for clustering, the result precision would be about 60% compared to 98% in case of visual clustering. After clustering, they label the clusters. For labeling, they use the frequent itemset mining algorithm which gives speed and scalability. In our case, we use statistical approach to find label candidates.

After they have found the label candidates for their clusters, web search is ran to detect Wikipedia articles. From every result they match the pictures of the article with the pictures of the cluster. If the pictures match, they connect the Wikipedia article with the cluster. Having a link to Wikipedia, they will have an annotation to the clusters which can be used in different ways, for example to get the name of the object.

The article is dealing with the very same topics we are interested in. They have described all the steps in the pipeline of labeling the picture clusters. We are not using visual comparison of pictures as proposed in their article, nevertheless the methodology mentioned is very interesting and as can be seen, there are similarities in our approaches. Some points which are worth looking into to improve our results: distinguishing between objects and events, looking into different textual similarity measures. In addition, we do not have clusters of pictures in the same sense they propose. In our case, the same picture could be used for several objects.

Assigning geographic coordinates to objects (e.g. pictures) is a popular topic. In [50] the authors present different ways to select terms which help to assign coordinates to Flickr photos and Wikipedia articles. One of the methods they mention is geographical spread (originally presented in [9]), which is used to find location-relevant terms. The idea is very similar to what we use in our method to find phrases which are candidates for tags. They look whether a term refers to a precise area (landmarks such as Eiffel Tower) or to a broader region (country of France). We use the similar idea, but do it country wise and aggregate the results over all the countries.

In [50] the goal of using the mentioned geographical spread is different. They want to locate tags which are location-specific. A picture with a tag which gives as much information about the location as possible helps to assign coordinates to the picture. Consider "Eiffel Tower": if the given tag is on the picture, it is relatively easy to assign coordinates to the picture. In contrast, if a picture has a tag "beach", assigning the exact location to the picture based on that tag is prone to errors.

Authors of [42] tackle a similar task - they want to assign tags to geotagged photos. Their approach is different: using existing tags from nearby pictures. For each photo they will look at other photos which are located close to the given photo and use the tags from those nearby photos. In addition they apply methods like visual similarity, different users employing the tag etc. to filter out suitable tags. In our methodology we take the near-by photos and use titles to extract tags. The idea to filter out some of the photos which may add noise is promising. In the work presented in our paper we have not applied filtering, but use a threshold to accept only phrases with a high enough appearance rate.

In [35] the authors automatically build a database of geographic landmarks. They use Wikipedia and Panoramio as the data sources and employ web search to identify the names, categories, coordinates and ranks for geographical objects. For names they use Wikipedia articles and links to other pages. In addition to Wikipedia, they extract information from the Panoramio titles. Given a coordinate, they find pictures within the specified distance. Then they extract capitalized words which are adjacent to the geographical concept (e.g. Museum).

Similarly to our approach, they use the first sentence of Wikipedia to extract the type for the object. For the objects which do not have a Wikipedia article, they extract the type from the candidate name which matches with a geographical concept: this will be the first candidate for the object type. A web search for the object name is executed and the geographical terms are collected from the top results' web pages. Extracted terms will be type candidates for the given object. Given type candidates are compared with search queries combined with the object name. The candidate which gets the most of the search results is assigned as the type for the object.

In our approach we do not use a fixed list of geographical terms as type candidates. Instead we learn the possible candidates from the picture titles and use additional crowdsourced portals to extract possible types for the objects. Usually there are several different categories which apply for an object, therefore our approach finds a set of tags.

There is a lot of work done in the field of visual categorisation to detect type or the object in the picture or assign geographical location based on the contents of the picture. In [21] the authors use visual folksonomy to obtain tags for visually similar pictures. Pictures with user generated tags are extracted for image features. For non-tagged pictures the tags from similar images are used. In [20] the authors classify landmarks for large-scale image collection using visual, textual and temporal information. They show that if combined together, textual tag analysis improves the quality of other approaches. They use tags from Flickr to extend the feature vector for a picture. The authors show that tag-

based textual models outperform models which use only image features. In our case we do not use photo tags: instead we make use of the photo titles, which add noise to the data.

2.4 Recommendation system

Recommender systems have grown out from filtering. The goal of filtering is to be able to find relevant information in large databases. A recommender system should take into account the user preferences along with other criteria, like opening times, distance from other objects etc. in tourism domain [39].

Recommender systems can be divided into two groups. One uses tourists' historical travel behaviour (content-based approach). The other is based on the history of the tourists who are similar to the current user (collaborative filtering approach). For travel recommendation using only content-based or collaborative filtering approach limits the capabilities [39]. Using content-based approach allows recommender system to suggest objects which the user has liked earlier. But it is very common that trip purpose and time and location can be different which makes it very hard to suggest objects in another city (business trip vs family trip). In order to do that, a very large amount of the user's history has to be available. The collaborative filtering can provide good results if there are users with similar preferred activities. As a trip is more complex than a book, using only other users' historic preferences does not always provide satisfiable results.

Research on recommendation systems is a very popular topic. A survey about recommendation system applications in tourism is provided in [4]. They provide suggestions on designing and developing a tourism recommender system like how to get more data about the user and her interests and how to use ontologies to structure the knowledge. Another survey of tourism related recommender system is provided in [6]. They have divided their work into sections based on the different subdomains within the tourism sector. A separate overview of hotel, restaurants, trip planning etc. is provided in the survey.

In [10] the authors propose a Bayesian network to estimate user's preferred activities. In their proposal the age, occupation and personality influence the user type. The user type along with the motivation affects the preferred activities' probabilities. For attraction ranking they use the persons' past travel behaviour. The user will be presented a map with proposed objects. The user can add/remove objects to give feedback about the recommended objects. The system also provides an itinerary along with distances, travel times and visit durations.

In [29] they introduce a recommender system which calculates a ranking function for every event in the database. The function takes into account the following restrictions:

- Proximity - the distance from the last activity location;
- Price - item must fit into user's budget;
- Time - item gets penalized if its time slot is outside user's preferred available time;
- Profile - similar items in user's previous trips or in current trip plan are found;
- Diversity - the plan has to be diverse, e.g. no point to send the user to the cinema twice.

The user can interact with the recommender in order to add, change or delete activities.

In [30] they allow users to specify the interest in specific topics in the scale $0 - 100\%$. The same concepts are used in their ontology which describes the objects in their database.

The next step is to specify demographic information like the origin of the user, with whom the user travels with, location of the accommodation, budget and the travel dates. The values for the questions are used to filter the results (by budget, by location). The selection of questions is based on a previous study with $30,000$ filled questionnaires. After the visit, the user can rate the objects in her plan. This feedback about different objects is taken into account in the next planning phase. After all the explicit information about the user is gathered, a list of activities (objects) are proposed to the user. The user can interact with the system in order to add the object into her plan, remove objects, see other similar objects etc. All those interactions are taken into account in order to propose additional objects.

The SigTur/E-Destination recommender system uses user specified interest scores in different topics and uses those for the corresponding ontology concept with the confidence value 100%. Based on the user actions, scores for the concepts can be change. The range for each concept score is between -1 and 1. If the user removes a certain object, the score for the given concept is lowered, and vice a versa. All the actions modify both the score and the confidence level for each concept. In addition to content-based recommendation, they use collaborative filtering to find similar users to recommend objects they have liked. They take into account user provided answers to about the interests and demographic questions to compare the users. In order to tackle the problem of low number of users, they have used $30,000$ questionnaires to find out how demographic values relate to interest scores. For example, elderly people usually like Relaxation concept. They also take into account user activities during the planning phase in order to find other users with similar actions.

The SigTur/E-Destination takes all the different aspects into account to calculate a preference value for each concept in the ontology, with a certain confidence level. As the ontology is hierarchical, the scores are propagates with specific rules between different levels. The concepts with a very low confidence score are discarded. Every activity will receive a preference score and a confidence score based on the concepts it has, averaging the values of the concepts associated to it. Activities which are outside of the preferred budget range are removed from recommendation. Both the score and the confidence level is used to order the proposed objects.

In [16] they use a genetic algorithm to find the best travel plan for the user. They start with random plans and then combine two plans into a better offspring. After a plan is generated, it is evaluated. During evaluation process, different properties give penalties. For example if the object is closed at the time of visit, the score for this object is 0. POIs are matched against user profile from the nine criteria corresponding to the parameter of focus and taste. Focus indicates functional features requested by the user for the trip. Taste refers to emotional characteristics of the trip. They evaluated their planner on real people and found that compared to manual planning, the saved time by the CT-Planner4 was evaluated as the highest benefit of the system. On the other hand, the quality of the plan was not highly evaluated, and 30% of the users said they value their manual plan better than CT-Planner4 provided plan.

The next version of the same project is described in [17]. In previous version they calculated score in different categories. If the POI had different categories, it ended up with a high score. Instead, if a POI has an high score in one category, this will have a high score for the matching user preference. The information about POIs is kept in Excel which has to be filled manually, which makes it a challenge to keep the data updated.

In [2] they use the previously mentioned CT-Planner recommender system, but this time add congestion information for POIs. They use Google's popular times data source

to get the crowd estimation for an object. This helps to provide POIs with possible and preferred opening times. For example, if the object is too crowded, it is not planned for the user.

Farokhi et al.[8] propose a system that employs multi criteria user-based and item-based collaborative filtering (CF) approaches. They use TripAdvisor dataset to experiment with their proposed methods. Fuzzy C-means algorithms along with k-means algorithms have been used to improve the accuracy of recommendation for user-based and item-based CF approaches. They acknowledge that the use of multi-criteria rating improves the recommender accuracy and provide more realistic recommendation that are close to users' interests.

In [5] the authors use data from Wikipedia to find POIs in the region. For each geo-referenced Wikipedia object they match nearby Flickr images. The count of images can be used to get the popularity of the object. They also use images from the same user to detect possible visit time of the object - taken into account the times of the first and the last image. In the experiments they showed how using information about a visit to one city can be used to recommend objects in another city.

Similarly to Brilhante et al., Lim et al.lim2015personalized use Wikipedia and Flickr to recommend personalised tours. In their work, they use a concept of time-based user interest which indicates the interest in a category based on the time she has spent at such a POI with the given category.

In [12] the authors describe a recommender system which takes into account user's check-in data, profile changes and information about friends from Facebook. In the case of the user "cold-start" problem where the system does not know enough information about the new user, relationships with friends can represent the user's interests.

Author in [38] give a comprehensive overview of how to design a recommender system. They propose a methodology for recommending objects for groups. They describe the creation of a group profile based on the profiles of each member. The experiments show that groups with the tourist of similar interests perform better than random or dissimilar groups.

Su et al.[46] propose a big data architecture to support recommendation of cultural content. They collect data from social networks to calculate popularity, mood and interest for each object. The system relies on a context-aware hybrid recommendation strategy that is deployed on a multi-layer architecture based on big data technologies.

Neidhardt et al.[31] propose a picture-based approach to deal with the recommendation. The pictures are used collect information for the user profile. In the evaluation phase they show that the feedback about the experience was really good - about 90% of the participants agreed that the experience was inspiring.

As travel recommendation is often personal and emotional, having an accurate recommendation algorithm can still yield in an unpleasant experience. There are several reasons to it. For example, Neidhardt et al.[31] show that users often do not trust the information provided by the system. Ricci et tal.[40] point out how recommender systems' goals extend beyond the accuracy of the algorithms used. For example, how the data is presented to the user, how the recommended items are explained. One of the key point is to have a quality source data to build the recommender system upon. As the data itself is changing rapidly (restaurants closed, new concerts happening etc.), acquiring the data often and automatically is a crucial element of the whole system.

3 Involvement in recommender projects

The research covered in the thesis took place during collaborative projects in ELIKO[1] with the participation of people from ELIKO, Tallinn University of Technology, several companies and organizations.

A distant, yet related starting point for our research was the EU project Smart Museum[2] which strived towards creating a recommender system / tour guide for museums, employing RFID tags attached to the seeworthy objects.

The Smart Museum project launched a research project at ELIKO with the aim of creating a more general content-based recommender system for tourism. This project resulted in the creation of three separate publicly available automated recommender systems we will briefly cover.

Sightsplanner

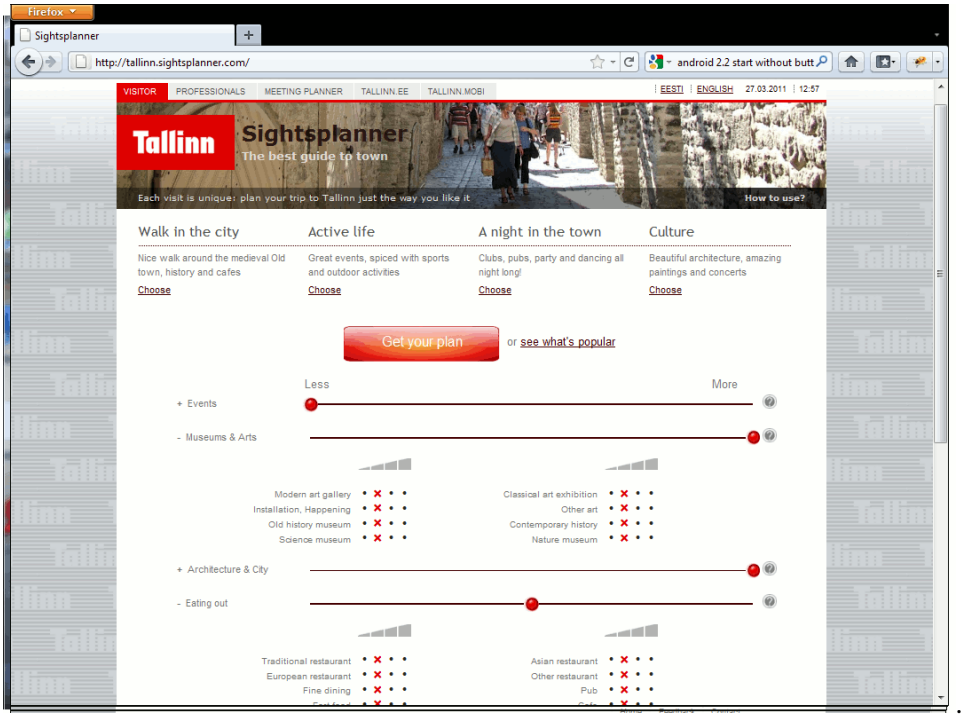

Figure 1: Preferences selection of the Sightsplanner system

The Sightsplanner system[3] (Figure 1) was an experiment in building an automated tourism recommender system for Tallinn, based on

- harvesting and merging available location data and content from various sources,
- providing a slider-based user interface for selecting personal interests,

[1] http://www.eliko.ee
[2] http://www.smartmuseum.eu/
[3] http://tallinn.sightsplanner.com, currently not deployed

- calculating best matches with the personal interests, using a probability-based ontology and a reasoning system.

Visit Estonia

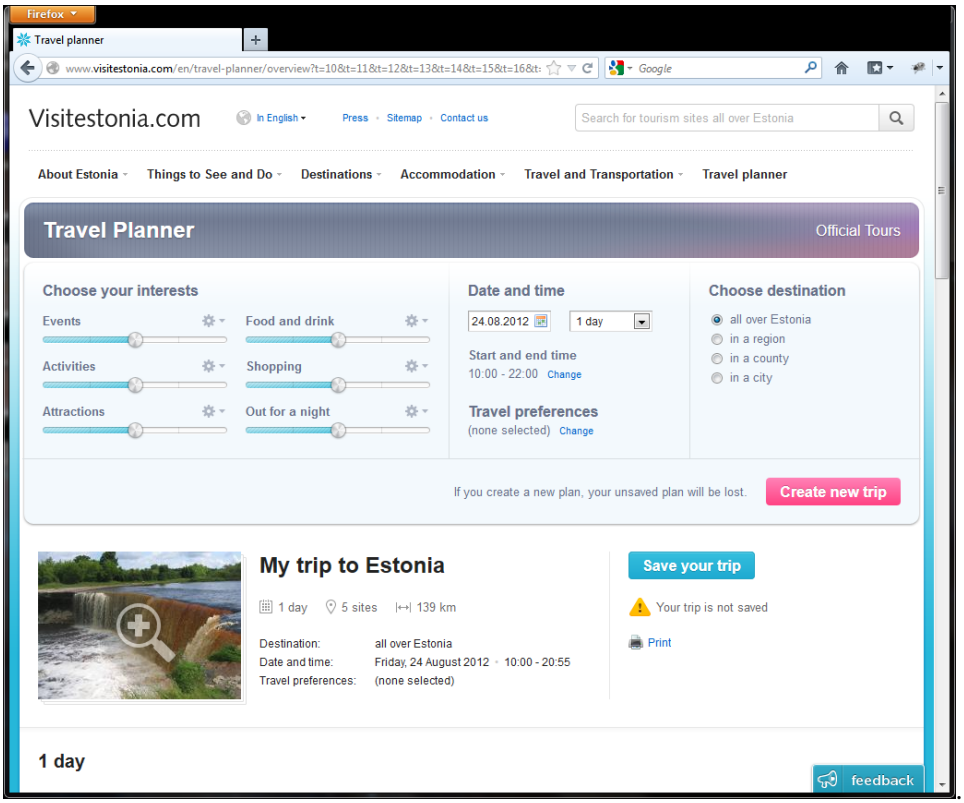

Figure 2: The main preferences selection of the Visit Estonia recommender

The automated recommender component of the main Estonian tourism system[4] (Figure 2) generalized the previous approach for the whole Estonia. The main differences from the previously mentioned Sightsplanner project are as follows:

- official location information from the Visit Estonia database was used as the main source of content,
- a slider-based user interface for selecting personal interests was reworked according to the specifics of the database,
- importance/popularity of locations was estimated and combined from both the content and usage statistics.

After the recent major update to the Visit Estonia portal the automated recommender component was not deployed.

[4] http://www.visitestonia.com

Sightsmap

Sightsmap[5] (Figures 3, 4, 5, 6, 7, 8) is a whole-earth recommendation system started at ELIKO and continued at the Tallinn University of Technology as a testbed for research in tourism recommenders. The system is deployed and regularly updated. It has a signficant number of users: Google Analytics counts the site having caȯne and a half million distinct users over its whole lifetime. The project has been featured in numerous large international publications.

Sightsmap takes a very different approach than the previous two systems. First and foremost, all the content along with the information about popularities of places of interest is crowd-sourced, i.e. stems from contributions of millions of individual people taking photos, writing or looking at Wikipedia pages and checking into Foursquare places. No manual curation has been applied to place selection.

- The focus is on generating a zoomable worldwide heatmap of (visually) interesting places to visit.

- The main data source is Panoramio[6] with ca 40 million user-taken photos with focus on tourism and interesting/beautiful places; we harvest the locations and tags of photos and use these as a basis for location popularity heatmap and location tagging.

- The concrete POIs (places of interest) and their names are selected for popular areas, based on additionally harvesting and merging Wikipedia[7], Wikitravel[8] and Foursquare[9] along with mining and combining the popularities according to these sites.

- POIs are enriched with names and tags created by merging the names from the beforementioned sources and the multilingual photo titles/descriptions harvested from Panoramio.

The goal of the sightsmap.com project (see [48] for the initial report) is to build a world-wide tourism recommender along with a database of the sightseeing popularity and types of concrete places (POIs) and wider areas in the world, using purely crowd-sourced data. By sightseeing popularity we mean the estimate of a number of people visiting the place and considering it as an interesting place for sightseeing, as opposed to popular places with no or very little potential for sightseeing, like hospitals, schools, gas stations, bus stops and airports.

Obviously, some of the abovementioned popular non-sightseeing places like schools and railroad stations may in some exceptional cases be sightseeing places as well: famous old colleges, Grand Central Terminal of New York, etc. Two separate important categories of objects in tourism industry - hotels and restaurants - are similarly ambivalent: on one hand, utilitarian and not necessarily a target or cause for travelling, on the other hand, a source of emotions and sometimes also an important motivation for travel.

[5] http://sightsmap.com
[6] http://panoramio.com of Google (discontinued by Google)
[7] http://wikipedia.org
[8] http://wikitravel.org
[9] http://foursquare.com

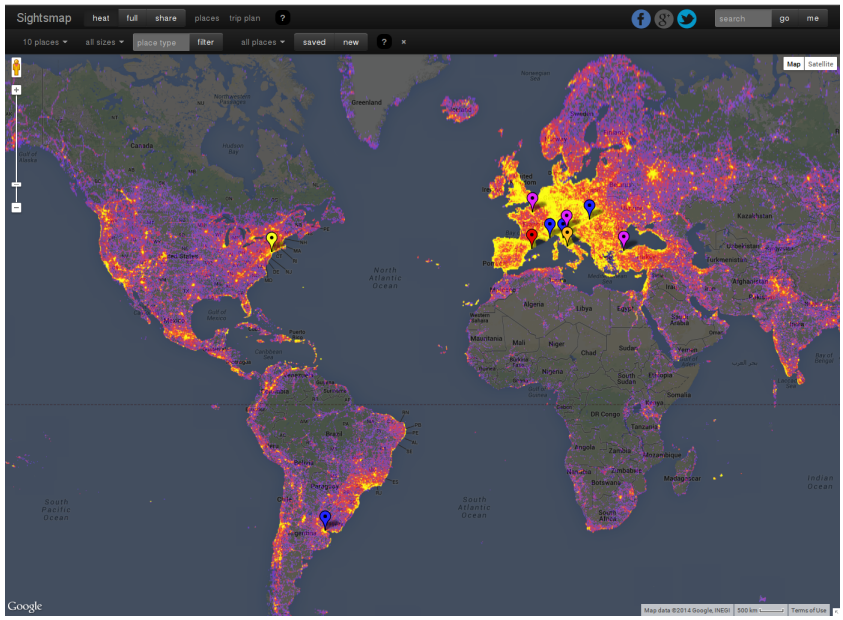

Figure 3: A screenshot of the heat map for most of the world on a single picture, with 10 top spots (1. New York, 2. Rome, 3. Barcelona, 4. Paris, 5. Istanbul) marked. Europe, especially the belt from Netherlands to Italy as well as the mountainous areas and the Spanish coastal areas dominate. In U.S. the mountainous areas in Utah and Colorado are well marked, in addition to coastal cities.

Our work is focused on popular sightseeing places regardless of their category. Hence we are not using data sources like TripAdvisor[10], Expedia[11], UrbanSpoon[12] or Zagat[13] which are primarily focused on specific categories, typically hotels and/or restaurants. Clearly, the hotels and restaurants are among the best crowd-described, -mapped, -reviewed and -rated tourism objects already.

The sightseeing popularity database we build is used by the sightsmap.com site for showing a zoomable and pannable touristic popularity heat map for any area in the world as an overlay on the standard Google maps. Popular areas on the map are labelled with an appropriate crowd-sourced name and colour-coded markers in the order of the relative popularity in the currently visible map area.

There are several advantages to using crowd sources is contrast to POI databases and guides created by experts in the tourism business. The crowd-sourced approach guarantees that there are no significant holes, i.e. interesting places and areas unmarked, and that the popularity estimates are relatively objective, which is hard to achieve by a small number of experts. Last not least, the popularity measurements can be done uniformly and comparably all over the world.

[10] TripAdvisor, see *http://www.tripadvisor.com/*
[11] Expedia, see *http://www.expedia.com/*
[12] UrbanSpoon, see *http://www.urbanspoon.com*
[13] Zagat, see *http://www.zagat.com*

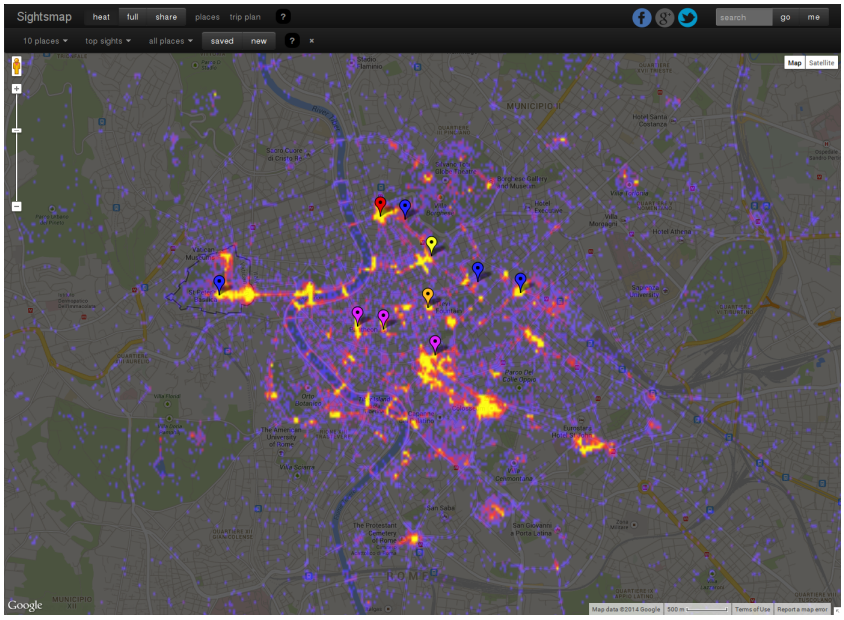

Figure 4: A screenshot of the heat map for Rome: the leading areas are Spanish steps / Trinita dei Monti, Trevi fountain, Piazza del Popolo, Pantheon, Trajan's column / Roman forum, Piazza Navona, St. Peter's Basilica, Palazzo Barberini, Piazza della Republica.

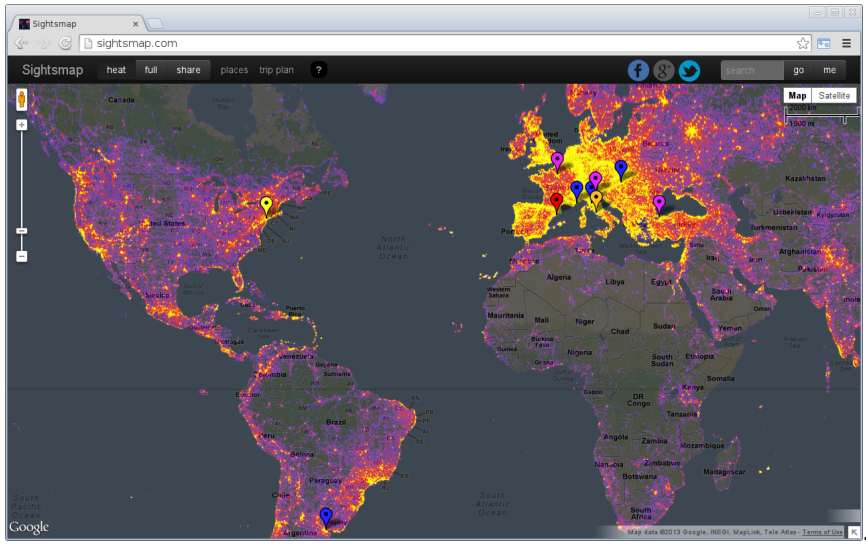

Figure 5: The whole earth heatmap along with topmost cities for tourism.

3.1 Recommender system toolset

Working with the different projects related to tourism recommender systems we have been able to formalize a toolset for building a recommender system. The document along with the attached publications describe the main parts in more details. Every step is important in order to be able to recommend objects to tourists. In this section we describe

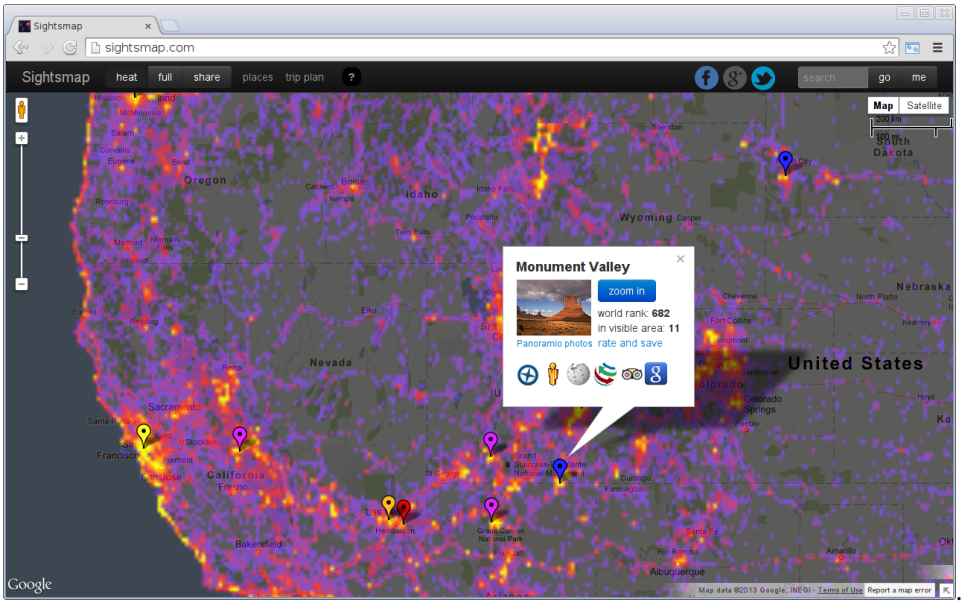

Figure 6: Western United States with the info window of one topmost area.

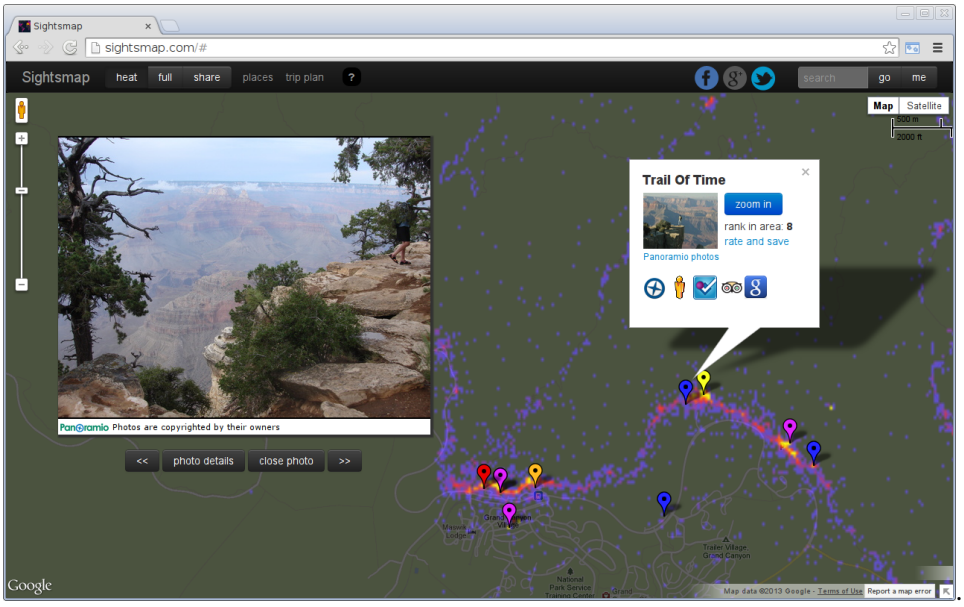

Figure 7: Zoomed in view of the main Grand Canyon walking trail.

how different parts fit into the whole system.

We start with gathering data about the tourism objects. We have implemented semi-automatic scraper tools which work with certain portals in order to gather information. For a new data source, some configuration has to be done in order to find the required data from the source. Gathering data from a database-like system usually means that the data has to be downloaded and then can be processed. Using regular web pages means

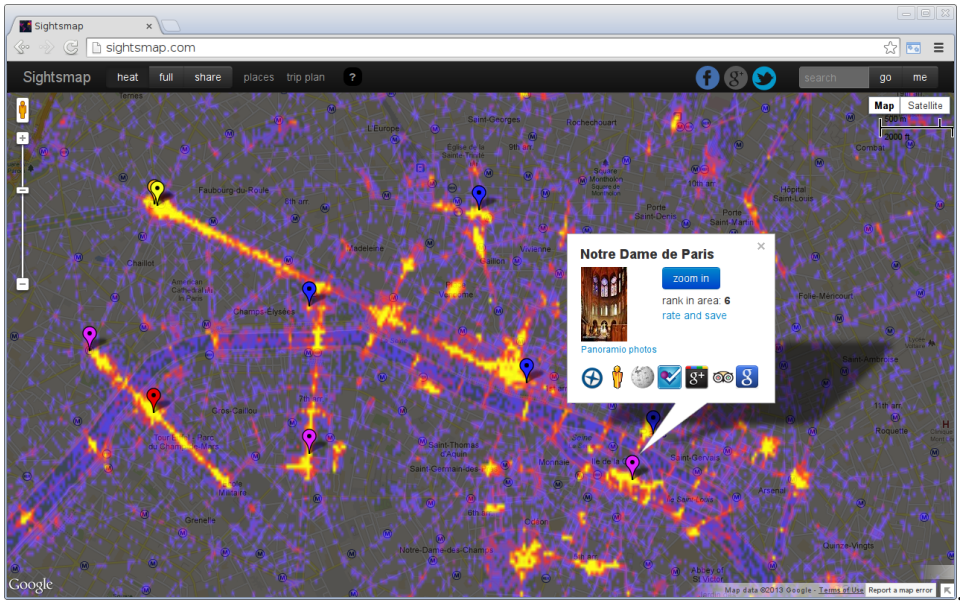

Figure 8: Heatmap of Paris with the most popular sites for photography.

there are a lot more issues which have to be solved first. For example: some pages are missing, data format is not strict to one structure etc. We have implemented a system which takes all this into consideration and is able to download data from different sources regularly.

Raw data from different sources has to be processed. There are several reasons to that. First is that the data can be structured incorrectly. Another is that some required parts of data can be missing. In addition, reliability issues may rise with certain data sources. We have introduced a system where missing information is automatically translated from another language. Also, data from different sources is merged and combined into cleaned up objects. A separate chapter describes the deduplication process which deals with duplicate objects from different data sources - how those are merged into one.

One of the data sources we use is a set of Panoramio pictures for the whole world. We have implemented a system which groups close-by images together and using only the titles of the images tries to suggest what is located at the given area. We have divided the work into two separate outcomes. The first is to find the categories for the given location. We have approached the problem with a more general goal: find appropriate tags based on the group of short texts (image titles in our case). Another problem is to find a title for this group. For those two tasks, we have used the same data (picture titles), but used different approaches. For tagging, we try to find more general tags which appear in a lot of countries. For naming, in the other hand, we try to find very specific tags which are not too general. We have taken different tagging frameworks and compared those with our proposed frequency based algorithm.

As the data itself is very important, it is equally important how to store the data. We have introduced an extended triple-store format which in addition to subject, predicate and object holds information about fact's reliability or strength, time the fact added etc. Storing data in the described format gives an opportunity to use it with a reasoner. We have described how the facts with the confidence score can be used to present tourism

object's types and how those can be combined with the user's profile. Score calculation for tourism objects is a totally different approach compared to widely popular collaborative filtering. In our case, we can provide interesting travel plans without any knowledge about the history of the user or tourism objects. Also, profile can be changed for each trip, which makes it a flexible option to use in recommender system.

4 Data acquisition and information extraction

In order to provide accurate and up to date recommendations, the information about the tourism objects have to be updated regularly. We have implemented a web scraping system which gathers data from six different web portals all with different data structure. For each data source we have manually described the structure of the page. The scraper gathers the data and transforms it into our own ontology. We have divided data into more static and more dynamic objects. One-time happenings (events, concerts etc.) are more rapidly changing and need to be updated regularly (once a day). Other objects (places of interest, monuments, parks etc.) have more static information which does not change for months or even for years. Based on the category, we scrape certain data sources with a different regularity. Usually a data source has information of either about events or places of interest, so each data source has its own update schedule.

A brief overview of the architecture of data importer is presented in Figure 9. The system can connect to several data sources using manually created scraper algorithms. Each scraper downloads the data from the source using predefined structure configuration. The data is then transformed into our structure and sent to the Merger. The Merger functionality is described in Chapter 5.

Each object in our database has several facts which are described with the following fields: *property*, *value*, *language*, *datatype*, *score*, *source*, *timestamp*.

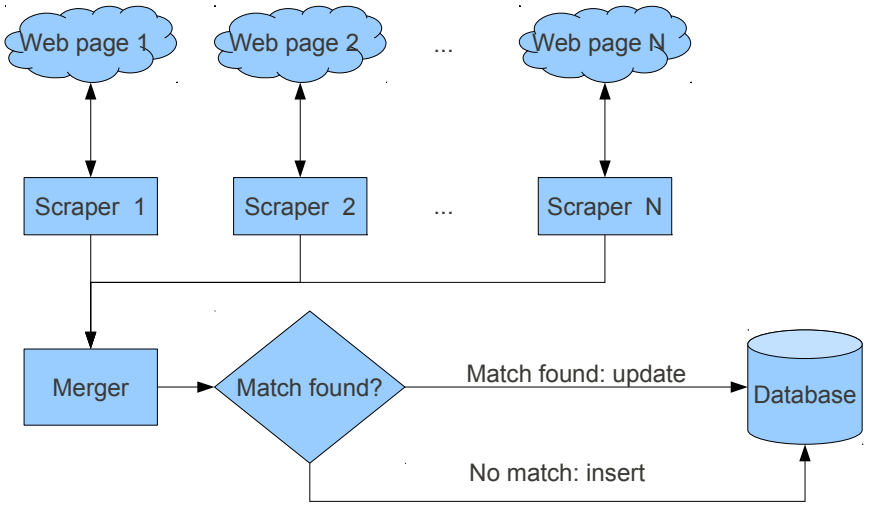

Figure 9: Architecture of data scraping system.

Although our methods focus on detecting sightseeing popularity, the notion itself is ambiguous and contains several different subcomponents: visual attractiveness, general public awareness about the place, the number of actual physical visitors etc. Each of the data sources used covers some components better than the others, effectively complementing each other. The data sources have been harvested using the public web API-s for Panoramio (www.panoramio.com/) and Foursquare (https://foursquare.com/) or downloaded in the already converted semantic format: Wikipedia (http://en.wikipedia.org) is

downloaded in the form of the DBpedia RDF database, later complemented with the public Wikipedia logfiles. Harvesting and downloading has been performed during 2012 and 2013.

- Our main data source Panoramio.com represents the visual component of sightseeing: something beautiful or interesting to see. Panoramio contains ca 44 million geotagged photos uploaded by users. For several reasons, the Panoramio photos are dominated by the touristic and sightseeing interest, in contrast to more private photos on Flickr (http://www.flickr.com/). Google maps and Google earth (http://www.google.com/earth/) use the Panoramio photos as their photo layer. We have downloaded only the metadata - location, photographer, title - not the actual photo files.

- The second data source Wikipedia represents the general public awareness about the place. We could safely say that all the interesting places, historic events, people etc. with public interest above a certain threshold do have a Wikipedia article. Places and historic events are normally geotagged in Wikipedia. The popularity - the exact number of readings in a selected time period - of each Wikipedia article can be obtained from the publicly available logfiles. We are using ca 700,000 geotagged Wikipedia articles with types which do not indicate noninterestingness for touristic purposes (like articles about plants, animals, people). We use full logfiles for two days, one selected from summer, the other from winter.

- The third data source Wikitravel (http://wikitravel.org/) essentially complements Wikipedia. Places above a certain touristic interestingness threshold normally have a Wikitravel article corresponding to some Wikipedia article. We are using the list of existing Wikitravel article names to detect whether a Wikipedia article has a complementing Wikitravel article as well.

- The fourth data source Foursquare gives an estimate of the number of people actually visiting the place. A large percentage of visits and a large percentage of Foursquare places are used and created by the local people visiting offices and eating lunch. Foursquare, differently from all the above sources, has a fairly detailed and well-used system for the crowd-sourced typing of places. We have downloaded not the whole Foursquare places database, but only ca two million places, preferring the places Foursquare presents when asked for a circle around some of the top hotspots we have previously found out from the analysis of the Sightsmap photos. We harvest several concentric circles around each place previously determined to be visually popular enough: small circles for objects in the cities and large circles outside or around the cities. In the other words, we have only downloaded the more popular Foursquare places in the neighbourhood of the more visually popular (world-wide) places.

4.1 Heat map generation, labelling and data merging

The heat map generation has two separate outcomes. First, it generates the visual heat map overlays for the map. We use the browser-based Google maps as the underlying map. Second, it generates a detailed popularity data for each small rectangular area (a pixel on the heat map) for each zoom level, which is later used for labelling, harvesting additional information etc. The heat map generation is done separately for six different zoom levels of the world, each with each own granularity. Additionally, the seventh layer is

a set of high-resolution heat maps, each typically covering one city, created for ca 15,000 top spots in the world. The resolution of these high-resolution heat maps depends on the popularity rank of the hotspots: the more photos, the higher the resolution, up to the street level for the top 500.

Our algorithm takes into account both the number of photos and the number of separate photographers in the Panoramio database for each area. The colour of each pixel on the heat map is calculated by a logarithm-like root function, different for each zoom layer. We use one byte for the colour information, with the the top popular places being bright yellow, followed by orange, red, purple and blue hues.

4.1.1 Wikipedia labels

The pure visual popularity heat map lacks a clear indication of what exactly is there in a hot area. In short, the top spots in each view have to be marked and the markers should ideally contain the name and the pointers to the most relevant information about the places.

Our basic solution for creating these markers, finding the titles and providing pointers is to look for a most popular geotagged Wikipedia article at or very close to each top hotspot at each heat map grid. Articles with an obviously unsuitable type (like plants, animals, and people) are excluded. This method guarantees that, for example, on the whole-world view where each hot spot pixel corresponds to a relatively large area, we automatically get the Wikipedia city articles as the most popular, but as we zoom in, the area for each pixel becomes smaller and we will start getting markers and articles about villages, beaches, castles etc.

The actual algorithm is the following. First we cluster the heat map dots to avoid showing lots of markers very close to each other. Then we look for the most popular Wikipedia articles near the hotspots: the higher-ranked a heat map spot is, the larger the area to search. If nothing is found or the found article has a much lower popularity than the heat map spot, we do not attach anything to the hotspot. Otherwise we connect a hot spot to the Wikipedia article plus the corresponding Wikitravel article, if available.

In order to generate the popularity data and a popularity-sorted list of Wikipedia articles we use the Wikipedia logfiles plus an additional coefficient giving a significant bonus to Wikipedia articles with a type suitable for sightseeing, for example, world heritage sites.

It is worth noting that knowing a highest-ranked Wikipedia article for an area helps users to google for more, since the article always gives us a title of the place to look for.

5 Data deduplication (using machine learning)

Gathering data from different sources yields in duplicate objects in the database. The same object is present in different sources and therefore ends up several times in our database. Having duplicate POIs on the map or in the recommended trip plan is something the system should avoid. One possibility would be to remove duplicate entries and leave only the unique ones. Instead, we merge information from different sources into one prominent object. We also have introduced a source quality score. Certain data sources are more reliable than others. Each fact about the object (name, description, address etc.) is stored as a separate fact in our database. Along with the record a reliability score is stored. The score is in the range 0 - 1. 1 indicates a certain fact whereas 0 indicates a certain false fact (those are not stored).

When duplicate objects are found, the data is merged into one representative object. To minimize the complexity, we do not keep duplicate facts about the objects. So, each final object has one name, one address etc. In order to do that, all the facts from the original objects are scanned through. For each distinct field, only the most prominent fact (the fact with the highest score) is kept.

The goal is to have all the information available in English. Most of the data sources we used in our work were available only in Estonian. We have introduced an automatic translation for certain fields. The translated texts have usually a lower confidence score. In cases where there is a text in English available, this is used. In cases where the text is missing, the translated text is used.

We gathered data from Estonian tourism sites about restaurants and events. We have presented a general algorithm to calculate object similarity in publication V. The system has different comparator functions which take two values and return the similarity between the values. The values can be texts (names or descriptions), addresses (latitude, longitude), addresses (textual representation of street, building, city), starting time etc. For two tourism objects, several such comparator functions are applied and combined into overall similarity of the objects.

As events in the database have starting time whereas restaurants do not (we do not count opening times here), we do not compare events to restaurants. In order to have a more general approach, we have introduced a so called "check function". Each comparator can have a check function which goal is to indicate whether the given two objects are comparable at all. In our case, start time comparator check function does not allow comparing restaurant with an event. Those helper functions can eliminate possible duplicate candidates and limit the search space.

Different comparators used are: location, event start time and text comparators. The latter is used for textual fields, title and description for example. The text is divided into keywords and keywords are compared. Text comparator usually has a lower importance compared to other comparators. The location comparator takes into account both geographical coordinates (latitude and longitude) and address information. Some objects only have address, some only have geocoordinates. With coordinates we calculate distance between two points. For some large objects, the same shopping centre can be 200 meters off compared to different sources. In addition to the coordinates we use textual address when comparing location. We split street address and building number and compare those separately. We also remove certain common keywords like "st", "street", "rd", "road" etc.

Location comparator has the highest importance. The same title (for example McDonald's) can exist in different places, therefore the address or the geocoordinates indicate the actual difference (or sameness). In case the location matches, totally different title

should indicate a different object. For example restaurants in the same shopping centre - although the address is the same, the title should indicate the difference. Taking all this into account, we have started with importance 0.15 for the title. The threshold for the duplicate objects is 0.9 - if the total similarity score is 0.9 or higher, the two objects are marked as one. The location and other comparators can yield to 0.85 score (if the address and type match), then if the title is different, the objects end up with the score 0.85 which is below the threshold 0.9.

To illustrate the comparison process and results of the calculations, we provide two examples.

1) We have two objects:

Table 1: An example 1 of comparing objects

Property	**Object A**	**Object B**
#title	McDonald‘s Rocca	McDonald’s Rocca
#latitude	59.4258329	59.4258492
#longitude	24.65024	24.65464
#address	Paldiski mnt 102	Paldiski mnt. 102

When we compare locations by coordinates, we get 248 meters as the distance between the objects. Comparing addresses says it is the same building, which means that the location similarity is 1.0. If we had just used the coordinates-based calculation, the similarity would have been 0.0. The similarity between titles is also 1.0. Therefore, the total similarity is 1.0, which means the objects represent the same physical object.

2) We have two objects:

Table 2: An example 2 of comparing objects

Property	**Object A**	**Object B**
#title	The Little Prince	Carmen
#latitude	59.4346093673	59.4341302
#longitude	24.7507912449	24.7506081
#address	Estonia pst. 4	Estonia pst 4
#start_time	19:00	19:00

In this example there are two events at the same place. Both events take place in Estonian National Opera at the same time, but in different rooms. First, if we compare the coordinates, we get the distance 54 meters. This would give the location similarity 0.5 (100 meters would be 0.0, 0-10 meters would be 1.0). Again, we need to compare addresses (which are the same) and get the actual similarity 1.0. When comparing start times, we also get the similarity 1.0. The only comparable difference presented in our example is the title. As there are no matching words, the title similarity will be 0.0. We have selected the title comparator importance so that in the current situation it would affect the total similarity enough to fall below the threshold. The real numbers in our implementation are: threshold 0.9, title comparator importance 0.15 (with 0.0 title similarity, the maximum total similarity can be 0.85, which would be lower than the threshold). In addition to mentioned comparators, we compare event types (ballet *versus* opera), event homepage etc. The total similarity between those two objects is actually 0.7.

In order to improve the deduplication, we have set up a machine learning process to train a model for detecting duplicate objects. The next section provides a more formal representation of our problem.

5.1 Deduplication formulation

We define some notations which we will be using later.

Object A_i is a duplicate of object A_j if they represent the same physical object. Let d be a symmetric function which returns 1 if all its arguments are duplicates, 0 otherwise:

$$d(A_1,\ldots,A_n) = \begin{cases} 1 & \text{if all } A_i \text{ are duplicates} \\ 0 & \text{otherwise} \end{cases}$$

In addition to comparing different objects, in our notation an object is a duplicate of itself: $d(A,A) = 1$ or $d(A) = 1$.

A group of duplicates, called a *cluster*, is a set C which consists of at least one object from all the objects **O** so that all the included objects are duplicates:

$$C = \{A_1,\ldots,A_n | d(A_1,\ldots,A_n) = 1\}$$

A cluster could also consist of only one object $C = \{A\}$, as $d(A) = 1$.

A maximal group of duplicates, called a *maximal cluster*, is a cluster which cannot accept any new objects so that the completeness is not broken (there is no additional object which is a duplicate of the objects in the cluster).

In this paper, we are interested in *maximal clusters*. Therefore, we use term *cluster* to denote *maximal clusters* if not noted differently.

A similarity between two objects A and B is defined by the function S which is weighted average over similarity values:

$$S(A,B) = \frac{\sum_i w_i * sim_i(A,B)}{\sum_i w_i}$$

sim_i is a function which compares certain property or properties of two objects and returns a similarity score in range [0; 1]
w_i is a weight value for the similarity function sim_i.
The value of S will be in range [0; 1].

Two objects are duplicates by similarity function if the similarity function S between the objects A and B exceeds a threshold T:

$$S(A,B) \geq T \Rightarrow d(A,B) = 1$$

5.2 Experiments

Instead of manually trying to adjust the weight parameters of similarity function in definition 5.1, we have used machine learning to find the best settings. The section is divided into subsections about learning problem, data used for experiments, feature selection, sample selection, learning setup and results.

5.2.1 Learning problem definition

In our approach we compare two objects and find a similarity score based on the property similarity functions. Objects are compared pair-wise, and we have presented pair counts in Table 3. Our learning task is to separate positive (duplicate) and negative (non-duplicate) pairs.

Given the similarity function $S(A,B)$ (definition 5.1) we define $\mathbf{f_{A,B}}$ as a feature vector with f_i being the evaluation of sim_i on objects A,B.

$$\mathbf{f_{A,B}} = \langle f_1, f_2, \ldots, f_n \rangle$$

where

$$f_i = sim_i(A,B)$$

sim_i may be chosen from a set of similarity functions Sim.

A sample is given by $\mathbf{x_{A,B}} = \mathbf{f_{A,B}}$

A sample set

$X = \{\mathbf{x_{A_i,A_j}} | A_i, A_j \in \mathbf{O}\}$

A classification label is given by

$$y_{A,B} = \begin{cases} 1 & \text{if } d(A,B) = 1 \\ 0 & \text{if } d(A,B) = 0 \end{cases}$$

A label set

$Y = \{y_{A_i,A_j} | (A_i, A_j \in \mathbf{O}), (\mathbf{x_{A_i,A_j}} \in X)\}$

Label set Y has labels for the same object pairs which are present in the sample set X in the same order.

A sample $\mathbf{x_{A,B}}$ is called a positive sample if $y_{A,B} = 1$, e.g. object A and B are duplicates. If $y_{A,B} = 0$, then the sample $\mathbf{x_{A,B}}$ is called a negative sample.

The goal is to find a function which predicts whether two objects are duplicates and would yield in the highest prediction accuracy.

A sample in our learning setup is a set of similarity values between two objects. Every value in that set indicates a similarity of a certain property or properties between two objects. For example, if there are objects A and B, which are very similar, then the sample can have values $\mathbf{x_{A,B}} = 1.0, 0.9$. Another pair A and C might not be similar and have values $\mathbf{x_{A,C}} = 0.3, 0.1$. As every pair can have two possible outcomes, our learning problem is a classification problem: a pair which represents a duplicate has a label 1, a pair which represents a non-duplicate objects has a label 0.

5.2.2 Data for learning

Our training data consists of mostly eating places in Tallinn from 5 (five) different sources. For testing we have used a dataset from Riga which is not limited to eating places, but contains also museums, galleries etc. Testing our trained model on a different dataset from a different city gives us information whether our solution of duplicate detection can be applied for cross-city and cross-category datasets.

In order to have labelled samples, we have created a simple application where pairs can be marked as duplicates. Tallinn data contains about 1800 scraped objects, comparing every object to every other object would yield int 1.6M comparisons. In order to limit the number of comparisons, we only mark duplicates pairs (everything else is non-duplicate). For each object, we find top candidates for possible duplicates and manually mark the duplicate candidate(s). The candidates are offered based on the distance and type and also

Table 3: Statistics about the dataset for Tallinn and Riga

Property	**Tallinn**	**Riga**
Object count	1808	3839
Different sources	5	2
Non duplicates	478	3762
2-object groups (object count)	203 (406)	75 (150)
3-object groups (object count)	133 (399)	1 (3)
4-object groups (object count)	68 (272)	-
5-object groups (object count)	43 (215)	-
6-object groups (object count)	5 (30)	-
8-object groups (object count)	1 (8)	-
Positive (duplicate) pairs	1543	78
Negative (non-duplicate) pairs	1.6M	7M
Positive pair %	0.1 %	0.001 %

source is considered - there is no point to show candidates from the same data source. If all the pairs are marked, duplicate objects can form groups. If objects A and B are duplicates and objects B and C are duplicates, then objects A, B and C form one group.

Manual annotation still raised many questions amongst users who had to find duplicate pairs. Even if you are local and know most of the tourism objects, there are still cases, which cannot be solved with 100% confidence. Also, as we mentioned, we used candidate selection, which might have left some actual duplicates out. Objects, which were not in the candidate list, were not checked by the annotator. Altogether we believe that the error of manual deduplication can be about 3–5 % based on the feedback of the annotators.

We have presented an overview of the dataset in Table 3. For Tallinn dataset, we have total of 1808 scraped objects from 5 data sources. 478 objects did not have any duplicate objects (or we could say they form up 478 clusters each consisting of only one object), 406 objects formed 203 groups with 2 duplicates in each cluster, etc. As can be seen, some objects have duplicate entries in the same data source. For example, in Tallinn dataset there is one cluster which is merged from 8 initial objects (duplicate object was present once in 2 data sources and twice in 3 data sources). Total number of unique duplicate pairs is 1543. If object A is a duplicate of object B, then it is counted only once. All other possible unique pairs (about 1.6M) between the objects are non-duplicate pairs.

5.2.3 Feature selection

We have implemented about 20 different functions for features. The most important ones are:

- Title comparison using Levenshtein distance[14];
- Custom title comparison with weighted words (common words weigh less and therefore do not change the outcome too much);
- Custom title comparison, which we will describe below;
- Euclidean distance using originally scraped coordinates;
- Address string comparison;

[14]Levenshtein distance, edit distance, see *http://en.wikipedia.org/wiki/Levenshtein_distance*

- Euclidean distance using coordinates which were calculated from address strings.

For our custom comparison we match words from the titles of the objects under comparison. Every matching word gives a positive score, whereas non-matching word gives a negative score. Scores are added and normalized into range $[0, 1]$. Small difference is allowed when matching words. Titles are compared in every language and the best match is used as the outcome.

In addition to mentioned custom title comparison, we also have so-called custom title comparison with a join. Title is split and then consecutive words are joined into one and then the matches and non-matches are found as with previously described procedure. For example, if we have titles "McDonalds" and "Mc Donalds", then the regular custom title comparison would return 0 as there is no matching words, whereas after joining two words in the second name we would compare the same titles, and the result will be high score.

5.2.4 Sample selection

As we presented in Table 3, the possible number of object pairs for Tallinn data is about 1.6M. The table also shows that only 0.1% of those are positive samples (duplicate pairs). If we would take all the samples, then we would have several problems:

- Generating a feature value set for 1.6M pairs takes time;
- Learning with a large number of data takes a lot of time;
- The balance between positive and negative samples is heavily skewed.

One of our goal is to reduce the number of samples. If we take randomly 10,000 samples for Tallinn data, we would only have about 15 positive training samples, which is obviously too few. Instead, we take all the positive samples and negative samples which have high similarity values (near-duplicate objects). Of course, we cannot leave out the low negative samples (all feature values near to zero), otherwise we might end up with classification which only recognizes mid-values as non-duplicates. About 1,000 samples have close to zero feature values. For Riga dataset, we have taken more samples. Riga data is used only for testing and it has about 100,000 samples: 78 positive ones, about 10,000 low negative ones, the rest is mid-valued or near-duplicate negative samples.

5.2.5 Learning setup

We have used Python software scikit-learn[15] to assist our learning process. The software supports various number of different learning algorithms. For our problem, we have used SVM (Support Vectore Machine) classification and decision trees. For SVM, we used grid search, which tries several different parameters and returns the one with the best results. The grid search tries both linear and radial basis function (RBF) as a kernel. For decision trees, we are using extra trees which train several (30 in our case) independent models randomly and uses average over the models to predict. The both learning algorithms are run with all the combination of all the features. This way we can find out the most important features. We also would like to minimize the calculation costs for prediction, therefore we try to minimize the number of features necessary for a model.

As mentioned earlier, we use Tallinn data for training and testing, Riga data is used only for evaluation. Tallinn dataset is divided into two equal sized parts where the ratio of positive and negative samples also remains the same. One part is for development and the

[15] scikit-learn: machine learning in Python, see *http://scikit-learn.sourceforge.net/stable/*

Table 4: Learning results using one feature

Feature (code)	Algorithm	Tallinn			Riga		
		precision	recall	f-score	precision	recall	f-score
Address string comparison (ADD)	rbf	0.92	0.94	0.93	0.52	0.91	0.66
	extree	0.96	0.92	0.94	0.59	0.88	0.71
Title comparison without joining words (T1)	rbf	0.99	0.85	0.91	0.76	0.91	0.83
	extree	0.99	0.85	0.91	0.65	0.91	0.76
Title comparison with joining words (T2)	rbf	0.98	0.89	0.94	0.69	0.96	0.80
	extree	0.98	0.92	0.95	0.52	0.97	0.68
Title comparison with edit distance (ED)	rbf	0.96	0.85	0.90	0.09	0.94	0.16
	extree	0.97	0.86	0.91	0.14	0.91	0.24
Distance with original source coordinates (OD)	rbf	0.76	0.67	0.71	0.65	0.67	0.66
	extree	0.84	0.82	0.83	0.36	0.90	0.51
Distance with re-calculated coordinates (RD)	linear	0.93	0.95	0.94	0.55	0.96	0.70
	extree	0.96	0.95	0.95	0.71	0.88	0.79
Title comparison without joining words, with word weights (TW1)	linear	0.97	0.90	0.94	0.36	0.99	0.53
	extree	0.97	0.96	0.96	0.27	0.99	0.43
Title comparison with joining words, with word weights (TW2)	linear	0.97	0.94	0.95	0.28	1.00	0.43
	extree	0.98	0.97	0.98	0.28	0.99	0.44

Table 5: Learning results using two features

Feature codes	Algorithm	Tallinn			Riga		
		precision	recall	f-score	precision	recall	f-score
ADD + T1	rbf	0.99	0.95	0.97	0.93	0.99	0.96
	extree	0.99	0.97	0.98	0.73	0.99	0.84
T1 + RD	linear	0.98	0.97	0.98	0.93	1.00	0.96
	extree	1.00	0.97	0.98	0.93	0.99	0.96
RD + TW1	linear	0.97	0.99	0.98	0.71	1.00	0.83
	extree	1.00	0.98	0.99	0.88	1.00	0.93
RD + TW2	linear	0.99	0.97	0.98	0.80	1.00	0.89
	extree	1.00	0.99	0.99	0.89	1.00	0.94
T2 + RD	rbf	0.98	0.99	0.98	0.86	1.00	0.92
	extree	1.00	0.98	0.99	0.94	0.99	0.96

Table 6: Learning results using two features

Feature codes	Algorithm	Tallinn			Riga		
		precision	recall	f-score	precision	recall	f-score
ADD + T1 + RD	rbf	0.99	0.97	0.98	0.97	0.99	0.98
	extree	1.00	0.97	0.99	0.93	0.99	0.96
ADD + T2 + RD	linear	0.99	0.98	0.99	0.94	1.00	0.97
	extree	1.00	0.99	0.99	0.93	1.00	0.96
T1 + RD + TW1	linear	0.99	0.97	0.98	0.91	1.00	0.95
	extree	1.00	0.98	0.99	0.91	1.00	0.95
T1 + RD + TW2	linear	0.99	0.99	0.99	0.89	1.00	0.94
	extree	1.00	0.99	0.99	0.94	1.00	0.97
T1 + RD + T2	rbf	0.98	0.99	0.98	0.87	1.00	0.93
	extree	1.00	0.98	0.99	0.96	0.99	0.97
ED + RD + T2	linear	0.98	0.99	0.98	0.91	1.00	0.95
	extree	1.00	0.99	0.99	0.92	0.99	0.95

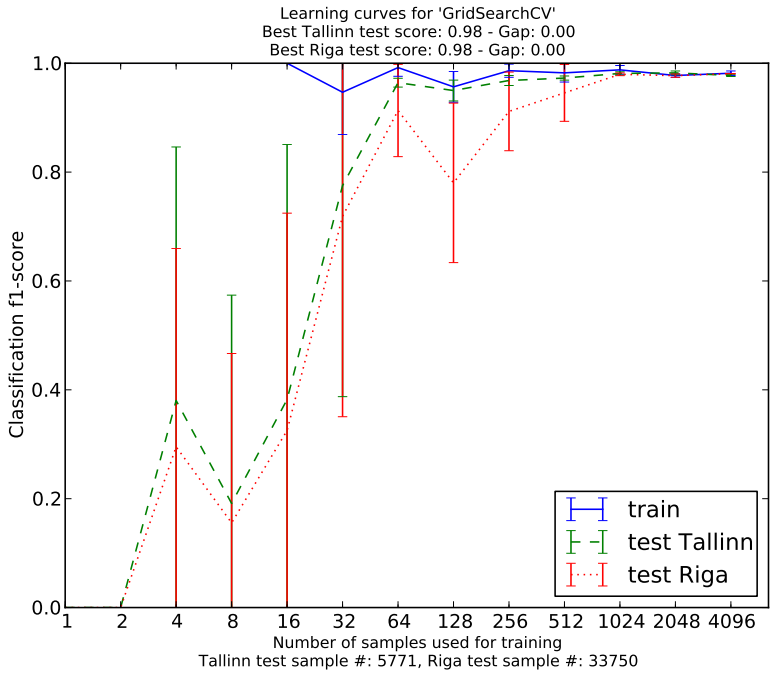

Figure 10: Learning curves for training, Tallinn test and Riga test data using grid search (SVM parameter optimization) and features ADD + T1 + RD

other is for evaluation. On development part, we do training with 10-fold cross-validation. The model which yields in best results on cross-validation, will be used for evaluation both with remaining Tallinn data and with Riga data.

5.2.6 Learning results

We have constructed several different datasets (different number of samples and different selection of samples) which were trained with different learning algorithms. The total number of test runs is over 400. Here we present the results for some of those tests. The tables described in this section all have the same structure. Each table has different number of features used for the learning problem. The first column indicates, which features are used for training and testing. Every feature set is evaluated with 2 different learning algorithms. The first one is Support Vector Machine (SVM) and the other is extra trees (extended version of decision trees). For SVM, we have shown the used kernel (linear or rbf - radial basis function). All this data is trained with the same amount of randomly chosen samples from Tallinn data. The ratio of positive and negative samples remains the same for training and testing data. All the tables present results which are trained with 50% of Tallinn data (5,771 samples, 771 of those are positive). The other part of Tallinn data will be used for evaluation. In addition, we have evaluated every trained model with Riga data (33,750 samples, 78 of those are positives).

Metrics used for evaluation are:

$$precision = \frac{tp}{tp + fp} \tag{1}$$

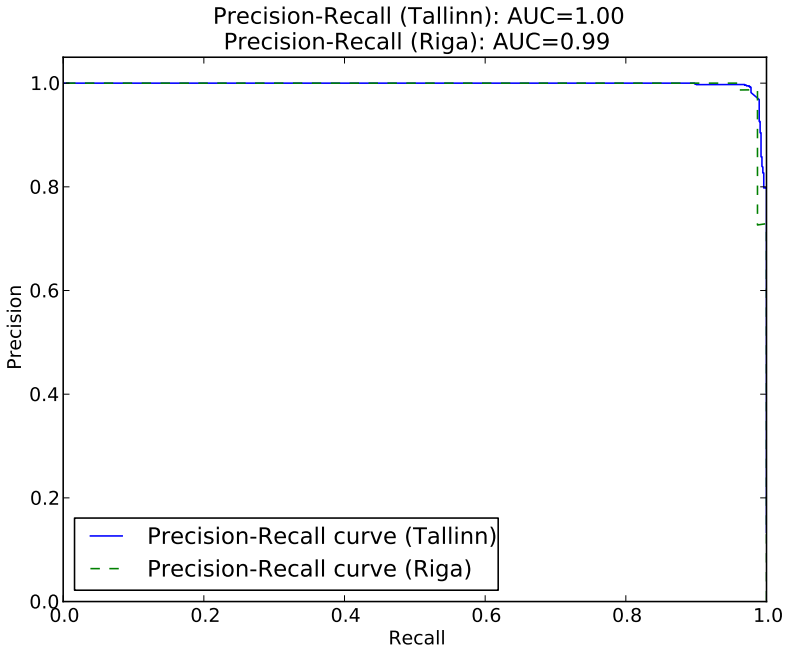

Figure 11: Precision-recall curve for features ADD + T1 + RD trained with SVM

$$recall = \frac{tp}{tp + fn} \tag{2}$$

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3}$$

where tp is the number of *true positive* predictions (how many predicted positive samples are actually positives), fn is the number of *false negative* predictions (predicted negative, but actually are positive), fp is the number of *false positive* predictions (predicted positive, actually are negative). In our problem, we concider both precision and recall equally important, therefore we use F_1 score[16] as the main evaluation of our model.

The Table 4 shows the results for 8 best features used alone. For training and testing, only values of one feature (for example title similarity) was used. The codes after the names are used in other tables to point to concrete feature. From the results we can see that using only one feature, we can have F-score near 0.8. We can also see that edit distance or Levenshtein distance (feature ED) alone does not separate duplicates from non-duplicates very well.

The Table 5 present results for some combination of two features. The F-score here is already very promising. We can see here that title similarity combined with location/address similarity gives good results. For example ADD + T1 (address string similarity and custom title similarity) and T1 + RD (custom title similarity and re-calculated distance similarity) both give F-scores about 0.95. We did test also with the combinations of three features which are presented in Table 6. All presented results have F-score near or over 0.95.

[16] F_1 score treats both precision and recall equally important

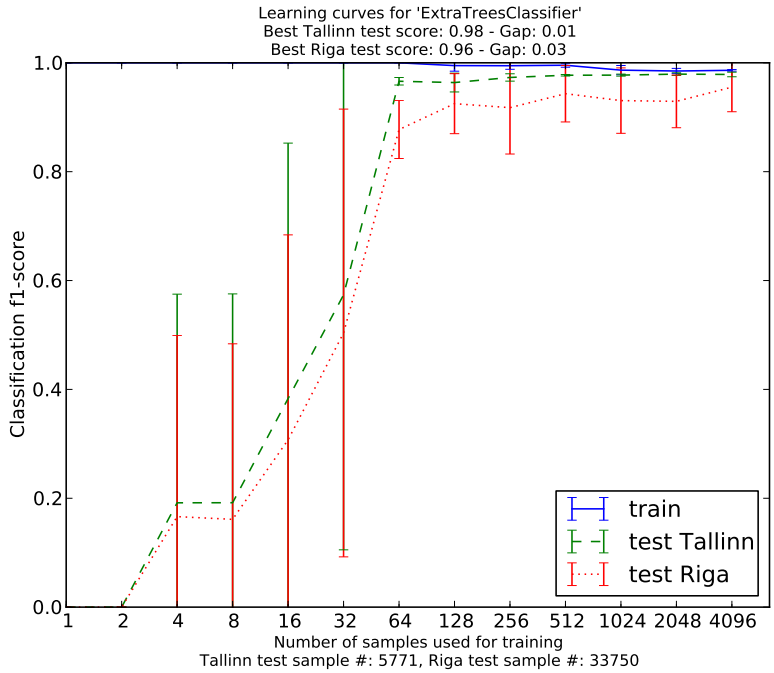

Figure 12: Learning curves for training, Tallinn test and Riga test data using extree (decision trees) and features ADD + T1 + RD

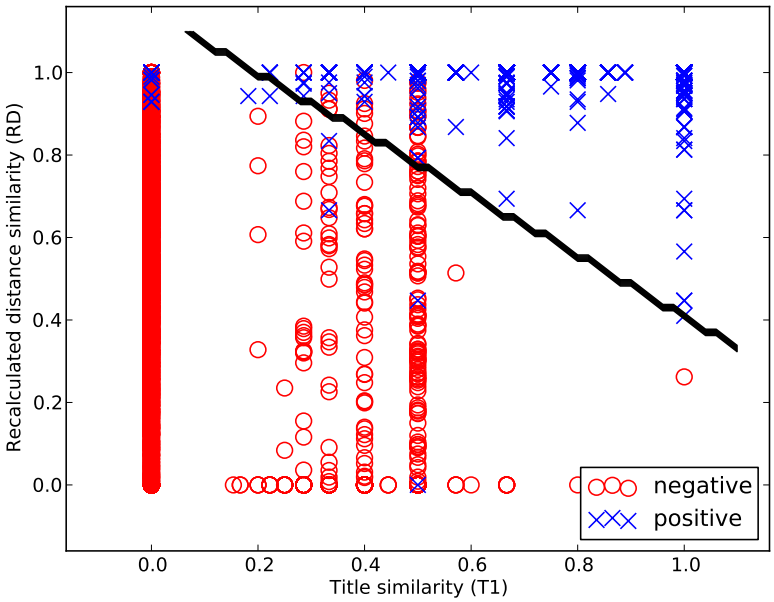

Figure 13: Tallinn dataset with features T1 + RD

The first feature set ADD + T1 + RD (address string similarity, title custom similarity and recalculated distance similarity) has the highest Riga test result. Also, as we stated earlier, we believe that the manual annotation of duplicate objects has also an error of 3–5 %, then 0.98 F-score is a very good result. We have completed tests also for 4 and 5 feature combinations, none of those performed better than 3 feature combination ADD + T1 + RD.

In addition, we evaluated the learning capability of our best model. In the Figure 10 we have drawn learning curves (which indicate F-scores) for training and testing datasets for different training dataset size using grid search (SVM). To give a better overview of the changes to the curves, we have used logarithmic scale. X-axis shows the training dataset size. Our results in the tables were trained with about 5,000 samples. As can be seen from the learning curve, we get good predictions already starting from 64 training samples. The learning curve also shows that our learning problem does not suffer from overfitting (our training accuracy is not too high compared to testing accuracy, the gap is close to 0 starting from 1,000 training samples).

If we compare the grid search learning curve to the decision trees' curve in Figure 12, we can see, that decision tree takes more training samples to get better result on Riga (additional test) data. In addition to learning curve, we have plotted precision and recall curve in Figure 11 for SVM model with features ADD + T1 + RD.

In the end, we will give an example plot of our samples with 2 features in Figure 13. We have used SVM to plot features T1 + RD (custom title similarity and recalculated distance similarity) which gives an idea, how the features are located. The black line is the trained class separator: right upper corner is for the positive (duplicates) predictions, lower left corner is for the negative (non-duplicates) predictions.

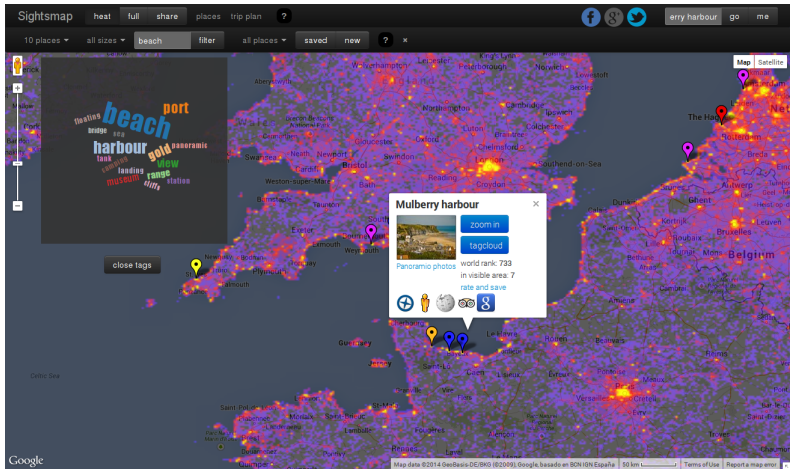

Figure 14: A screenshot of the selection by the "beach" tag in the English Channel area with a tag cloud for the Mulberry harbour area.

6 Location category and name detection

In order to enable searching objects and areas by type, we attach descriptive tags to several hundred thousand most popular objects. Tags are calculated from photo titles and in some cases augmented with a Foursquare type of a place.

The actual usage of tags in the sightsmap.com web application is twofold. First, a user can click and see a visual tagcloud of any important marker, giving an immediate rough idea of what are the interesting aspects of this particular place (Figure 14). Second, a user can start typing a word into the filter-by-type box, with autocompletion built in to dynamically present a list of all the matching tags, categories and their generalizations. Selecting a tag or a general category filters out all the locations where this tag or tags under the selected category have an importance over a certain threshold. We are satisfied with the usability of the described setup.

6.1 Multi-language titles

Before analysing a picture title, we attach a country code to every picture, using shape files from Thematic Mapping API[17]. This dataset has only the main borders of the countries, plus bigger islands. For some pictures, the surrounding country cannot be determined: in those cases we have found the closest border and assigned the picture to that country.

Knowing the country code for a picture gives an opportunity to distinguish languages. Although a lot of titles are in English, there are local tourists who use their own language to describe the picture. Due to a large size of the data set we have not used language detection as this is expensive. Knowing a country code also helps to process the picture titles in order to use country-based filters or properties. For example, consider a popular phrase present in a certain percentage of picture titles inside one country. This popular phrase might be something general (beach, museum) or the name of one specific object (Eiffel Tower). If the pictures with the given phrase are drawn on the map, we can see the coverage area for the given phrase in a country. In case the pictures with a certain phrase cover a relatively large area - compared to the whole country area - then the particular

[17] http://thematicmapping.org/downloads/world_borders.php, provided by Bjorn Sandvik

phrase is probably a general tag and could be considered as a type of the object. If the coverage area is rather small - for example, in case of Eiffel Tower only certain region of Paris is covered - we could say that this phrase matches a specific object: typically it is the name of the object in the pictures.

Phrases which appear in different countries all over the world indicate a general tag for an object (museum, beach, church). For example, a word "church" is present in titles in at least 150 countries. Eiffel Tower appears a lot in France, but is not used much in other countries.

We use automatic translation for popular phrases for every language. Since the translation process for large quantities is expensive, we select about 500 to 1,000 most popular phrases for a country. All of those phrases are then translated into English using Google Translate. After the translation process we merge phrases from different countries and from different languages. There are about one thousand different phrases which are popular in the photo titles all over the world.

6.2 Missing titles and types

For approximately seventy percent of the most popular areas we are able to find a local Wikipedia article or a Foursquare location providing both a likely name for the area and a likely type. In the Wikipedia case we use the first sentence from Wikipedia to extract type information. For example, the Wikipedia article about Stonehange has the following first sentence: "Stonehenge is a prehistoric monument in Wiltshire, England ...". The first sentence is often in a form "object is/was/are/were type" where we try to extract the type part. Another example for Tallinn Town Hall: "The Tallinn Town Hall (...) is a building in the Tallinn Old Town, Estonia ...". As can be seen from the last example, the indicated type can be very general.

Since about one third of our top spots are not connected to Wikipedia or Foursquare, extracting type or the title from concrete article is not available. In the next sections we will look into different methodologies we apply to our data to enrich it and present a novel idea which directs us towards getting the types and the title of a top spot.

6.3 Phrase extraction

Our approach is to use photo titles to understand what does the photo represent. Typically the title contains a name of the object or provides a hint about the object type, like "mountain" or "big church" etc. There are certainly titles with non-informative names like "me in 2010", "this was so cool" etc. However, the majority of the pictures have meaningful titles.

For every top spot we select an area around it, with the size depending on the popularity of the spot. Every picture in this area is processed. The title of the picture is tokenised into lower case words. We ignore commas, full-stops etc. For every tokenised title we will find the n-grams for n from 1 to 4. An n-gram is combined by taking n consecutive words from the title. A simple example: given a title "A picture of Big Ben", we will end up with tokens: "a", "picture", "of", "big", "ben". All 1-grams are: "a", "picture", "of", "big", "ben". Followed by 2 and 3-grams: "a picture", "picture of", "of big", "big ben", "a picture of", "picture of big", "of big ben". And 4-grams are: "a picture of big", "picture of big ben".

All the pictures around the top spots are processed for n-grams. We then select the most frequent n-grams. If the phrase is present in at least 10 percent of the titles from all the pictures in the area, the phrase will be stored as a candidate. The candidates are then used to deduct titles and types for the objects.

6.4 Names versus descriptive tags

A photo about a concrete object will probably have a title with the object's name. People taking pictures of Eiffel Tower will usually mention it in their picture titles. Obviously, pictures which mention the Eiffel Tower are located mainly around the object itself. A phrase which occurs only on the pictures from a small area is assumed to indicate a name of the object.

A lot of pictures do not mention a name of the object, but indicate a general type or description of the object. If the user does not know the name of the object in the picture, she can still write a general description or a type. A phrase which occurs in different locations is likely to be a general descriptive tag. In order to find such tags, we look at how often is a certain phrase used and how wide an area does it cover - the wider the area, the more general / common the tag is.

Finding a phrase coverage for every country separately gives us more options to analyse the results and draw conclusions. As mentioned earlier, we have to tackle the language problem. For each country we can assume the language of origin and translate the phrases into English. There is no need to translate every phrase we find: instead, we translate just the top 1,000 phrases by occurrence. In addition to the occurrence we also check the coverage, i.e. how large a percentage of the country is covered by photos containing a concrete phrase.

After we have found the top phrases for every country, the results are combined to form the worldwide set of tag words. Only phrases which are present in several countries are used as descriptive tags. The descriptive tags will be then propagated back to the top spots. Every top spot has a number of tag candidates: however, only those which ended up being tags in the whole world context will be considered.

6.5 Experimental results for photo title analysis

In this section we will present some results obtained using the process described earlier. We note that for our experiments we have only taken into account one-word tags. Phrases with two or more words are not considered.

We have about 45 million geo–tagged pictures from Panoramio. In addition to latitude and longitude, ca half of the pictures have a title - for us, those three fields are the main source of data. Additionally we have the owner identifier and more metadata along with the main fields.

As already mentioned, we have grouped pictures together by the location. Based on the amount of pictures in the group, we have selected 175,000 of the largest groups as our top spots. One top spot should point to a concrete tourism object: to be more exact, a visually attractive object based on the pictures taken. We use Wikipedia, Foursquare and also Geonames to match the top spots in order to get additional information about the location. The main goals are to get the name (what is shown on the pictures) and the types of objects on the pictures.

We will focus on the steps towards finding types for the objects. More concretely, we extract descriptive tags for every top spot: these are later used to filter out types. From 175,000 top spots we end up generating 28 million word n-grams or phrases. As we focus on one-word phrases, most of those generated n-grams are single words. On average, every top spot gets about 150 phrases attached.

For every country we find the area which a phrase covers. We use simple bounding box for all the phrase locations. If the bounding box covers more than 30% of the country's area and the phrase is present in at least 0.05% of the pictures of the given country, the

phrase will be used as a country phrase. In total we have 90,000 country phrases. In our database we distinguish between 235 countries, which gives 380 phrases for a country on average.

A more indepth view of the process is described in Algorithm 1. The algorithm uses the following thresholds:

- $T_{freq} = 0.0005$ (0.05%) - a tag has to be present in that ratio of pictures in a country to be considered as a candidate;
- $T_{bbox} = 0.3$ (30%)- a bounding box of the pictures with the given tag has to cover the given ratio of the country's land area;
- $T_{country\_freq} = 0.3$ (30%) - the ratio in how many countries the tag has to be present (as a candidate, e.g. covers the previous requirements) to be counted as a global tag.

6.6 Comparison of tag recommendation methods

To evaluate our tagging accuracy we have taken UK photos and compared the results with other phrase extraction methods. As our initial data is a set of titles or text in general, we use different language / text analytics tools. We have used UK as the most of the titles are in English and we will not use translation.

The detailed information about the comparison can be seen in the publication I.

We have manually annotated 200 objects with different types and locations all over UK. We will use those to validate our results. In addition, we will be using 100 of those to train our own models to compare with pretrained models.

We have taken 100 globally most popular phrases (stopwords and only small area phrases are filtered out) and use those as our set of available tags (or categories).

We used two evaluation methods to compare the results. The first is using f-score Figure (15) and the other is using a visual representation of the classification (Figure 16)[18]. We use scikit-learn provided f-score calculation functionality. As we allow multiple tags (three) for one cluster, we are using f-score with a built-in micro-average calculation. Micro-average depends on true positive, false positive and false negative results. In addition, we have to define what happens if one of the comparable objects has fewer classes than the other. For example, the case where the tag recommender method only provides one or two, but our manually annotated object has three classes. The missing classes will set to empty. When comparing the results, the f-score will be low as empty does not match the annotated class. This could be changed depending on the goal of the classification. For example, if the classifier finds fewer tags, there could be no penalty.

The results are shown in Figure 17. Detailed discussion on the results are provided in the publication I.

6.7 Manual selection and categorization

Despite the large amount of processed titles, the number of repeatedly used distinct tag words is not high at all. There are just approximately nine hundred tag words which are used more than twice. The top 10 of the list with the occurrence count is shown in the Table 8. Notice that some of the words like "saint" are likely to stem from place names which the algorithms failed to identify as names, while some words like "near" are clearly not suitable for use as tags.

[18] See https://cs.ttu.ee/research/projects/sightsmap/

Data: Countries C, pictures P
Result: Popular global tags $tags$
$tags =$ new list for popular tags
$tag\_countries =$ new hash map (tag $\rightarrow$ country objects)
for $c \in$ *countries* C **do**
    $tag\_pic =$ new hash map (tag $\rightarrow$ picture objects)
    for $p \in$*pictures for country c* P_c **do**
        for $t \in get\_words(p_{title})$ **do**
            if $t \notin tag\_pic$ **then**
                $tag\_pic[t]$ = new list
            end
            `// add picture into the list by tag`
            $tag\_pic[t].add(p)$
        end
    end
    $popular\_tag\_pic =$ new hash map (tag $\rightarrow$ picture objects)
    for $t \in keys(tag\_pic)$ **do**
        `// on how many pictures does the tag exist`
        $tag\_freq = |tag\_pic[t]|/|P_c|$
        if $tag\_freq > T_{freq}$ **then**
            `/* translate tag depending on the country's language`
               `to English */`
            $tag\_tr = translate(t, c)$
            $popular\_tag\_pic[tag\_tr].addall(tag\_pic[t])$
        end
    end
    $sorted_tag_pic =$sort $popular\_tag\_pic$ by picture count
    for $t \in keys(sorted\_tag\_pic)$ **do**
        `// calculate bounding box`
        $bbox\_area = get\_bbox\_area(sorted\_tag\_pic[t])$
        if $bbox\_area/C_{area} > T_{bbox}$ **then**
            `/* this tag is frequent and covers wider area, will`
               `use this for the given country */`
            $tag\_countries[t].add(c)$
        end
    end
end
for $t \in keys(tag\_countries)$ **do**
    `/* in how many countries the tag was found popular enough`
       `*/`
    if $|tag\_countries[t]|/|C| > T_{country\_freq}$ **then**
        $tags.add(t)$
    end
end

Algorithm 1: Finding relevant tags

Table 7: The size of data we work with.

Panoramio pictures	45M
Top spots	175k
Top spots phrases	28M
Country phrases	90k
Countries	235
Top spots tags	2.8M
Different tags	944

In Figure 18 we have shown all the objects which have a "beach" tag based on the image titles. Each red dot represents a cluster of images, where the tag "beach" is in top 3 tags. Each such cluster is potentially a POI which is related to beach. The eastern coast is missing dots as the data we used to plot the map were cut of at the concrete longitude.

In comparison, the very bottom of the list of words used more than twice is: baobab 124, hippo 106, pirogue 87, beaches 81, camels 80, ace 71, impala 66, jebel 63, indies 41, monasteries 17, hippos 14, coated 12, stillwater 11, recumbent 9, behold 8. Notice that the list stems from photo titles and hence there are just a few words for visually uninteresting information like a shop, pub or a bar.

For practical use it makes sense to generalize the tag words under large categories. Obviously, some of the words like "near" or "sunset" are not really useful for categorization. All in all, we selected 179 suitable tag words from the top 900 of the full list. Clearly there is no sense in generating categories for which there are almost no instances. Since we used the word list without changes, it contains a few words with both a singular and plural version as well as some words like reka and kerk, for which the automatic translation did not find a proper English correspondence.

Analysing the statistics of the tag words led us to create the following small list of actually useful categories, organized as a shallow taxonomy. The category names in the following tree start with the capital letters: all the other are tag words. We give a number of occurrences in the tag word and the number of summarized occurrences in the category after the word/category name. The number of tags shown in the tree are collected not from the whole Panoramio dataset, but from the 15,000 top spots: hence the numbers are ca ten times lower than in the Table 8. The dataset of these top spots along with the extracted tag candidates are available at https://github.com/tammet/sightsmap. The whole tree and its subtrees are sorted by the number of word occurrences.

- Landmark (103622) :
  - Historic (23626) :
    - Castle (5806): castle (4138), rampart (708), fortress (696), walls (264)
    - Monument/Memorial (5107) : monument (2266), cemetery (1215), memorial (1119), statue (221), grave (193), mausoleum (93)
    - old (7540), museum (2830), royal (553), ancient (440), historical (391), ruin (370), historic (329), abandoned (260)
  - Church (20125): church (9152), chapel (2337), cathedral (1814), temple (1614), holy (1170), monastery (1131), mosque (702), basilica (446), shrine (359), templom (351), cami (296), catholic (285), kerk (282), pagoda (186)
  - Harbour (9741) : port (3799), boat (2074), harbor (1776), ferry (861), ship (649), jetty (299), wharf (283)

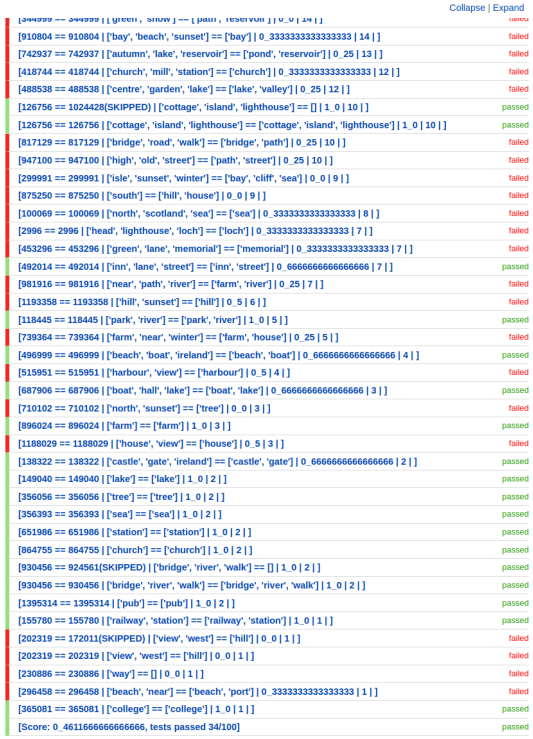

Figure 15: A screenshot of unit tests. Each row represents a test for one manually annotated cluster.

  - Education (5001) : school (2853), university (1110), college (548), library (490)
  - Palace/Villa (3528) : villa (2030), palace (764), ranch (434), manor (224), villas (76)
  - house (7324), bridge (6730), station (4845), tower (4814), plaza (2087), market (1989), hall (1555), dam (1522), fountain (1500), farm (1457), lighthouse (1385), mill (1321), airport (1283), palace (764), quarries (723), tunnel (671), stadium (585), residence (321), windmill (240), hut (215), chalet (213), bazaar (57)

- Nature (78943) :

  - Sea (18636) : beach (4858), island (2906), sea (2233), bay (2186), coast (2097), marine (1460), lagoon (990), ocean (638), surf (366), sailing (223), reef (218), coastline (180), yacht (156), fjord (123), beaches (2)
  - Mountain (13186) : hill (3886), high (2347), mountain (2200), mount (1351), peak (1070), mountains (666), cliff (459), cliffs (323), volcano (310), hills (302), glacier (272)
  - Garden (6254) : garden (3350), gazebo (881), flower (780), flowers (778), patio (465)
  - Lake (4711) : lake (2240), pond (1407), reservoir (665), lakes (399)
  - River (4194) : river (2307), creek (1358), stream (393), reka (115), fleuve (21)
  - Valley (3444) : valley (2818), canyon (562), wadi (64)

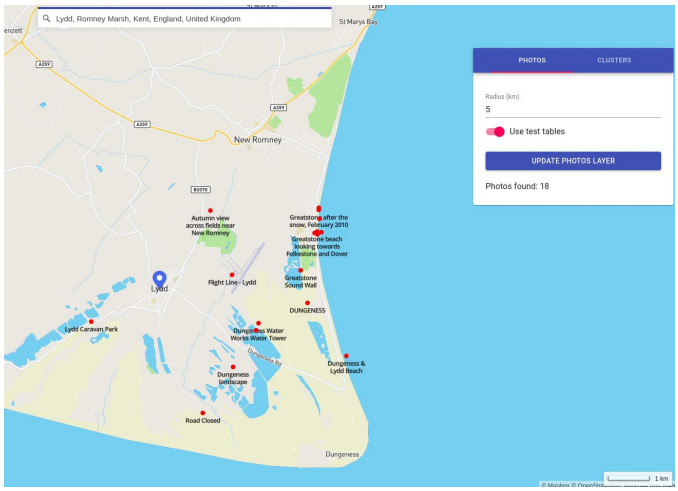

Figure 16: A screenshot of visual evaluation. The map can show each picture title individually, or the clusters and recommended tags. This way a larger set of clusters can be evaluated visually.

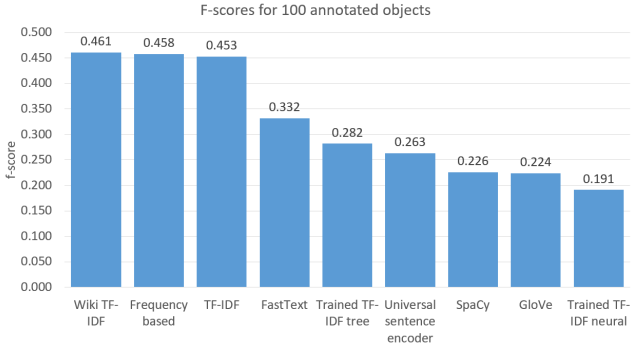

Figure 17: Results of all the different tagging methods.

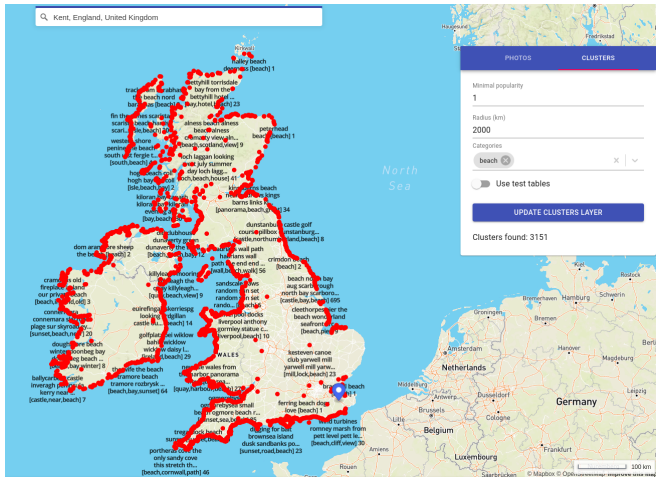

Figure 18: A screenshot of all the "beach" tags in UK based only on the image titles.

Table 8: Most popular tags all over the world.

Tag	# of pictures	# of countries
view	1122018	197
beach	560349	138
church	545210	147
park	437798	135
bridge	377299	126
river	368806	135
lake	348466	117
road	339388	175
castle	330983	66
sunset	306240	176
street	292761	139
hotel	283653	166
old	281107	158
port	274308	93
saint	269436	52
tower	255302	94
station	250416	129
house	238989	171
island	218904	126
city	199808	120
near	198220	187
home	188525	104

  - Waterfall (2579) : waterfall (1321), fall (811), falls (447)
  - Wildlife (912) : elephant (201), pelican (147), wildlife (135), crocodile (85), buffalo (75), camel (73), iguana (69), lizard (53), leopard (25), hippo (18), impala (14), camels (12), elephants (4), hippos (1)
  - park (8246), tree (2524), rock (2306), trail (1916), forest (1888), green (1641), trees (842), nature (738), shore (677), grove (610), rocks (519), dunes (453), caves (422), desert (413), cave (383), waterfront (352), palms (226), jungle (222), dune (202), cactus (181), safari (117), mangrove (85), baobab (33), paddy (31)

- View (19291) : view (13886), panoramic (1937), landscape (1579), beautiful (1394), skyline (299), panorama (196)
- Accommodation (7090) : hotel (6950), guesthouse (89), housing (51)
- Relaxation (6067) : club (1905), pool (1721), resort (1359), spa (788), casino (294)
- Food and drink (5378) : restaurant (1993), inn (1334), bar (1214), cafe (563), pub (274)
- Art (3059) : statues (1029), sculpture (563), theater (556), gallery (298), theatre (226), art (200), culture (187)

Table 9: The main table structure

Data type in PostgreSQL	Data type in memory DB	Description
integer	int	Unique row id
integer	uri	Object URI (reference to string table id)
integer	uri	Property URI (reference to string table id)
integer	depending on the data type	Value URI (reference to string table id)
real	double	Score/strength of given connection/association
integer	uri	Source URI of given fact (reference to string table id)
timestamp	int	Timestamp when this fact was created
datetime	int	Fact is valid starting from this datetime
datetime	int	Fact is valid until this datetime
integer	int	Whether this row is private (not visible) info

7 Data storage and object score calculation

7.1 Data storage

We have used two databases in our recommender systems: a relational database and an in-memory database. All the data manipulation takes place in PostgreSQL. All the deduction algorithms and scheduling for the recommender system takes place in the memory database. We have set up an automated synchronization for two databases. The information from PostgreSQL is imported into the memory database every night, since data about events are also scraped every night.

The database structure is based on the extended RDF model - the traditional triplet fields (*object*, *property*, *value*) are extended with meta-information about the triplet. The system has one main data table for records, all the values and URIs are stored in the second *string* table. The description of the main table can be seen in Table 9.

The *score* field shows the confidence or probability of the fact within range [0; 1]. For example, a fact can say that "church X has a name Oleviste" with the score 0.9. Another fact about the same object X may say "object X has a name Niguliste" with a lower score 0.5, intuitively meaning that the church X probably has a name "Oleviste".

The *source* field indicates the name of the source for the given fact. Facts come from different sources. Some are scraped from the web, some are manually inserted, others are reasoned from other facts etc. Different methods result in a different source name. For example, we often get different names for the same object from different web portals. Using source fields, we can make difference between the facts and favour one over the other.

The *timestamp* field shows the date and the time of the creation of the fact. The *validity start* and *validity end* fields indicate the period during which the fact is valid. Omitted values indicate the fact is always valid. The *privacy* indicates the visibility of the fact. The default value NULL means the fact is visible or public. Higher value means the fact is hidden (not visible) and cannot be used in certain situations.

The *object*, *property*, *value* and *source* fields all point to *string* table. The value in *string* table is kept as a string, therefore we need *datatype* field which indicates how the value in *string* table has to be treated. *Object*, *property* and *source* values are URIs. The *string* table only keeps one record for each URI - all the same URIs in the main table point to the same record in *string* table. The value of the *value* field depends on the datatype.

The in-memory database uses a similar structure. Most of the values are stored in the main table. Only long strings and URIs are stored separately and the pointer is stored in the main table.

The shared memory database has a built in reasoner to deduct new facts about the objects. We use RIF[3] style rules with an additional confidence score for each rule. The reasoner is mainly used to derive new types for objects.

An example of rules:
0.9: fact(?X type architecture) :- fact(?X type church).
0.9: fact(?X type drinking_place) :- fact(?X type bar).
0.7: fact(?X type eating_place) :- fact(?X type bar).

The first rule indicates that if the object has a type *church* with confidence score N then the object will have a new type *architecture* with score $0.9 * N$. Other rules indicate that a *bar* is also a *drinking place* (with a higher confidence) and an *eating place* (with a lower confidence).

Another use case for the reasoner is to set opening times for the objects which do not have those. The recommender system uses opening times to plan the timetable, therefore each object has to have opening times. Certain outdoor objects like monuments and statues are open all the time, while some are usually open during the day (10 am to 5 pm). Opening times are described either by weekday (for tourism objects) or concrete date (for events). The reasoner could also set opening times only for weekdays.

7.2 Score calculation

In order to recommend objects to a tourist a profile for the tourist is needed. One option is to use a long term profile with preferences over time. The tourist registers an account and sets her preferences about different interests. In our case, we have used single time preferences for the given planned trip. Depending on the purpose of the trip and the destination and other parameters, the preferences about the sights can vary. For example, a tourist could be interested in popular places in Paris. The second time she visits Paris, she would like to go hiking or see less popular objects.

The profile should indicate the interests of the tourist. In order to match the user preferences and object categories, we use the same ontology for both points of interests and user profile. That allows the system to match the same categories to calculate the scores for different objects.

Here is an example of a user profile:

```
user1 profile:
  music 70
  architecture 40
  museums 40
  sports 40
  food 50
```

The profile indicates preferences or interest for the given topics with the given scores. The score here is a value in range $[0, 100]$. Let's consider the following objects:

```
object1:
  music 50
  architecture 30
  museums 20
  food 10
object2:
  music 90
  sports 60
```

Objects have properties (topics) with confidence score. The scores are taken from the database where the *property* is *type*. The scores of the profile and the object are multiplied for each topic:

```
object1 for user1
  music 0.7 * 0.5 = 0.35
  architecture 0.4 * 0.3 = 0.12
  museums 0.4 * 0.2 = 0.08
  food 0.5 * 0.1 = 0.05
object2 for user1
  music 0.7 * 0.9 = 0.63
  sports 0.4 * 0.6 = 0.24
```

To get the whole score for the object, each topic score is multiplied by a weight and the results are summed up. The weight is $1/2^i$, where i indicates the index (order) of the given topic based on the multiplication value. The highest value is the first and has the index 0. So, the first weight will be 1, followed by: 0.5, 0.25, 0.125 etc. The idea of those weights is to not boost the score in cases where the object has a lot of types (compared to objects which have less types, but which match better). First few highest matches are important, the rest do not affect the score that much. The algorithm for score calculation is presented in publication VI.

After all the scores for the objects are found, a planning algorithm arranges object into an itinerary while trying to maximise the individual object ranking. The planning takes into account available opening times and tries to find diverse set of objects in different topics.

8 Conclusions

In the world containing a growing amount of data, providing the tourist with the right recommendations for her trip is important. This thesis develops both a methodology and several algorithms for extracting, integrating and selecting relevant information for building a structured database of POIs from a number of crowd-sourced databases like Panoramio and Flickr photobanks and Foursquare, Wikipedia and Geonames semi-structured place databases.

We compare our frequency based algorithm with the other tag extraction tools referred in the literature and show that the algorithm is significantly faster, yet gives the results with the same quality as the best pretrained models.

We have developed a tourism taxonomy based on the frequency of tags in the Panoramio pictures of the whole world. In addition to categories, we have developed an algorithm to find names for tourism objects around the world using Panoramio photo titles.

We develop a methodology for deduplication of POIs using machine learning and show that our algorithm improves the deduplication performance compared to a manually configured algorithm from 85% accuracy to 98%.

We have also developed a rule-based calculation algorithm using confidence scores to find the best match between a user profile and a tourism object. This match is used to indicate the interestingness of the object for the tourist.

The work has been carried out during the development of three different recommender systems: Sightsplanner for Tallinn, Visit Estonia for the whole Estonia and Sightsmap for the whole world. Sightsmap is still active, although unmaintained due to the closing of the Panoramio database by Google: the names, tags and categories shown on the map are a direct result of the work carried out in this thesis.

9 Future work

In our recent paper we showed promising results for a new algorithm of finding categories from crowdsourced Panoramio picture titles. For evaluation we have used only the pictures of UK. We are working on moving from UK to the whole world. As was pointed out in the publication, we have currently manually annotated 200 tourism objects in UK with tags. Manual annotation does give some indications on how the algorithms perform, but the size of our dataset is quite small. As we are broadening our area of focus, we have to find better means to get comparison data. In order to do that, we have started working on using Flickr tagged images. Images in Flickr have a title and geo-coordinates. This allows us to compare our tagging algorithm to be compared against user applied tags.

In addition to pictures, short texts like tweets from Twitter could be categorised. Tweets also have get-coordinates which potentially provides an opportunity to even find tourism related categories from tweets. People visit different places and tweet about those, information from the short text could be turned into tag clouds on the map. But in addition to tourism field, short text could be analysed for other type of tags or categorisation.

References

[1] Schema.org, 2020. [Online; accessed 2020-08-02].

[2] T. Aoike, B. Ho, T. Hara, J. Ota, and Y. Kurata. Utilising crowd information of tourist spots in an interactive tour recommender system. In *Information and Communication Technologies in Tourism 2019*, pages 27–39. Springer, 2019.

[3] H. Boley, G. Hallmark, M. Kifer, A. Paschke, A. Polleres, and D. Reynolds. Rif core dialect (second edition).

[4] J. Borràs, A. Moreno, and A. Valls. Intelligent tourism recommender systems: A survey. *Expert Systems with Applications*, 41(16):7370–7389, 2014.

[5] I. R. Brilhante, J. A. Macedo, F. M. Nardini, R. Perego, and C. Renso. On planning sightseeing tours with tripbuilder. *Information Processing & Management*, 51(2):1–15, 2015.

[6] K. Chaudhari and A. Thakkar. A comprehensive survey on travel recommender systems. *Archives of Computational Methods in Engineering*, pages 1–27, 2019.

[7] D. O. S. de Andrade, L. F. Maia, H. F. de Figueirêdo, W. Viana, F. Trinta, and C. de Souza Baptista. Photo annotation: a survey. *Multimedia Tools and Applications*, 77(1):423–457, 2018.

[8] N. Farokhi, M. Vahid, M. Nilashi, and O. Ibrahim. A multi-criteria recommender system for tourism using fuzzy approach. *Journal of Soft Computing and Decision Support Systems*, 3(4):19–29, 2016.

[9] C. Hauff and G.-J. Houben. Wistud at mediaeval 2011: Placing task. In *MediaEval*, 2011.

[10] Y. Huang and L. Bian. A bayesian network and analytic hierarchy process based personalized recommendations for tourist attractions over the internet. *Expert Systems with Applications*, 36(1):933–943, 2009.

[11] S. Kara, Ö. Alan, O. Sabuncu, S. Akpınar, N. K. Cicekli, and F. N. Alpaslan. An ontology-based retrieval system using semantic indexing. *Information Systems*, 37(4):294–305, 2012.

[12] K. Kesorn, W. Juraphanthong, and A. Salaiwarakul. Personalized attraction recommendation system for tourists through check-in data. *IEEE Access*, 5:26703–26721, 2017.

[13] J. Kim, M. Vasardani, and S. Winter. Similarity matching for integrating spatial information extracted from place descriptions. *International Journal of Geographical Information Science*, 31(1):56–80, 2017.

[14] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff. Semantic annotation, indexing, and retrieval. *Journal of Web Semantics*, 2(1):49–79, 2004.

[15] I. Koumarelas, A. Kroschk, C. Mosley, and F. Naumann. Experience: Enhancing address matching with geocoding and similarity measure selection. *J. Data and Information Quality*, 10(2), Sept. 2018.

[16] Y. Kurata and T. Hara. Ct-planner4: Toward a more user-friendly interactive day-tour planner. In *Information and communication technologies in tourism 2014*, pages 73–86. Springer, 2013.

[17] Y. Kurata, Y. Shinagawa, and T. Hara. Ct-planner5: a computer-aided tour planning service which profits both tourists and destinations. In *Workshop on Tourism Recommender Systems, RecSys*, volume 15, pages 35–42, 2015.

[18] A. Kuusik, E. Reilent, I. Lõõbas, and A. Luberg. Data acquisition software architecture for patient monitoring devices. *Elektronika ir Elektrotechnika*, 105(9):97–100, Jan. 2010.

[19] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[20] Y. Li, D. J. Crandall, and D. P. Huttenlocher. Landmark classification in large-scale image collections. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1957–1964. IEEE, 2009.

[21] S. Lindstaedt, R. Mörzinger, R. Sorschag, V. Pammer, and G. Thallinger. Automatic image annotation using visual content and folksonomies. *Multimedia Tools and Applications*, 42(1):97–113, 2009.

[22] I. Lõõbas, E. Reilent, A. Anier, A. Luberg, and A. Kuusik. Towards semantic contextual content-centric assisted living solution. In *The 12th IEEE International Conference on e-Health Networking, Applications and Services*, pages 56–60. IEEE, 2010.

[23] A. Luberg, M. Granitzer, H. Wu, P. Järv, and T. Tammet. Information retrieval and deduplication for tourism recommender sightsplanner. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, WIMS '12, New York, NY, USA, 2012. Association for Computing Machinery.

[24] A. Luberg, P. Järv, K. Schoefegger, and T. Tammet. Context-aware and multilingual information extraction for a tourist recommender system. In *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies*, i-KNOW '11, New York, NY, USA, 2011. Association for Computing Machinery.

[25] A. Luberg, P. Järv, and T. Tammet. Information extraction for a tourist recommender system. In M. Fuchs, F. Ricci, and L. Cantoni, editors, *Information and Communication Technologies in Tourism 2012*, pages 332–343, Vienna, 2012. Springer Vienna.

[26] A. Luberg, J. Pindis, and T. Tammet. Sights, titles and tags: Mining a worldwide photo database for sightseeing. In *Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics*, WIMS 2020, page 149–158, New York, NY, USA, 2020. Association for Computing Machinery.

[27] A. Luberg, T. Tammet, and P. Jarv. Extended triple store structure used in recommender system. In *2011 22nd International Workshop on Database and Expert Systems Applications*, pages 539–543, Aug 2011.

[28] A. Luberg, T. Tammet, and P. Järv. Smart city: A rule-based tourist recommendation system. In R. Law, M. Fuchs, and F. Ricci, editors, *Information and Communication Technologies in Tourism 2011*, pages 51–62, Vienna, 2011. Springer Vienna.

[29] A. Montejo-Ráez, J. M. Perea-Ortega, M. Á. García-Cumbreras, and F. Martínez-Santiago. Otiŭm: A web based planner for tourism and leisure. *Expert Systems with Applications*, 38(8):10085–10093, 2011.

[30] A. Moreno, A. Valls, D. Isern, L. Marin, and J. Borràs. Sigtur/e-destination: ontology-based personalized recommendation of tourism and leisure activities. *Engineering applications of artificial intelligence*, 26(1):633–651, 2013.

[31] J. Neidhardt, L. Seyfang, R. Schuster, and H. Werthner. A picture-based approach to recommender systems. *Information Technology & Tourism*, 15(1):49–69, 2015.

[32] K. Patroumpas, M. Alexakis, G. Giannopoulos, and S. Athanasiou. Triplegeo: an etl tool for transforming geospatial data into rdf triples. In *Edbt/Icdt Workshops*, pages 275–278. Citeseer, 2014.

[33] K. Patroumpas, D. Skoutas, G. Mandilaras, G. Giannopoulos, and S. Athanasiou. Exposing points of interest as linked geospatial data. In *Proceedings of the 16th International Symposium on Spatial and Temporal Databases*, pages 21–30, 2019.

[34] S. Paumier. Unitex 2.0. user manual, université paris-est marne-la-vallée, 2008.

[35] A. Popescu, G. Grefenstette, and P. A. Moëllic. Gazetiki: automatic creation of a geographical gazetteer. In *Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries*, pages 85–93. ACM, 2008.

[36] J. Pustejovsky, R. Ingria, R. Sauri, J. M. Castaño, J. Littman, R. J. Gaizauskas, A. Setzer, G. Katz, and I. Mani. The specification language timeml., 2005.

[37] T. Quack, B. Leibe, and L. Van Gool. World-scale mining of objects and events from community photo collections. In *Proceedings of the 2008 international conference on Content-based image and video retrieval*, pages 47–56. ACM, 2008.

[38] L. Ravi and S. Vairavasundaram. A collaborative location based travel recommendation system through enhanced rating prediction for the group of users. *Computational intelligence and neuroscience*, 2016.

[39] F. Ricci. Travel recommender systems. *IEEE Intelligent Systems*, 17(6):55–57, 2002.

[40] F. Ricci, L. Rokach, and B. Shapira. Recommender systems: introduction and challenges. In *Recommender systems handbook*, pages 1–34. Springer, 2015.

[41] M. H. Salas-Olmedo, B. Moya-Gómez, J. C. García-Palomares, and J. Gutiérrez. Tourists' digital footprint in cities: Comparing big data sources. *Tourism Management*, 66:13–25, 2018.

[42] A. Silva and B. Martins. Tag recommendation for georeferenced photos. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks*, LBSN '11, pages 57–64, New York, NY, USA, 2011. ACM.

[43] F. Soualah-Alila, M. Coustaty, N. Rempulski, and A. Doucet. Datatourism: designing an architecture to process tourism data. In *Information and communication technologies in tourism 2016*, pages 751–763. Springer, 2016.

[44] F. Soualah-Alila, C. Faucher, F. Bertrand, M. Coustaty, and A. Doucet. Applying semantic web technologies for improving the visibility of tourism data. In *Proceedings of the Eighth Workshop on Exploiting Semantic Annotations in Information Retrieval*, pages 5–10, 2015.

[45] E. Spyrou and P. Mylonas. Analyzing flickr metadata to extract location-based information and semantically organize its photo content. *Neurocomputing*, 172:114 – 133, 2016.

[46] X. Su, G. Sperlì, V. Moscato, A. Picariello, C. Esposito, and C. Choi. An edge intelligence empowered recommender system enabling cultural heritage applications. *IEEE Transactions on Industrial Informatics*, 15(7):4266–4275, 2019.

[47] T. Tammet and A. Luberg. Combining fuzzy and probabilistic reasoning for crowdsourced categorization and tagging. In *Web Reasoning and Rule Systems: 8th International Conference, RR 2014, Athens, Greece, September 15-17, 2014. Proceedings*, volume 8741, page 247. Springer, 2014.

[48] T. Tammet, A. Luberg, and P. Järv. Sightsmap: Crowd-sourced popularity of the world places. In L. Cantoni and Z. P. Xiang, editors, *Information and Communication Technologies in Tourism 2013*, pages 314–325, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[49] D. Tunaoglu, Ö. Alan, O. Sabuncu, S. Akpinar, N. K. Cicekli, and F. N. Alpaslan. Event extraction from turkish football web-casting texts using hand-crafted templates. In *2009 IEEE International Conference on Semantic Computing*, pages 466–472. IEEE, 2009.

[50] O. Van Laere, J. Quinn, S. Schockaert, and B. Dhoedt. Spatially Aware Term Selection for Geotagging. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):221–234, Jan. 2014.

[51] H. Wu, A. Luberg, and T. Tammet. Ranking domain objects by wisdom of web pages. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, pages 1–4, 2012.

[52] Y. Zheng, X. Fen, X. Xie, S. Peng, and J. Fu. Detecting nearly duplicated records in location datasets. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 137–143, 2010.

Acknowledgements

I would like to thank my supervisor, Professor Tanel Tammet, for motivation, guidance and support. I am also grateful to all of my colleagues at the Institute of Software Science who have supported my studies.

I want to thank the people in TU Graz, Austria where I was a research fellow in 2011–2012. I am especially grateful to Michael Granitzer who guided me during my stay and inspired me to tackle new scientific goals.

Most of my publications are published based on my work in Eliko. I want to thank the colleagues in Eliko who made all the projects possible and supported my research.

I am grateful to my children, Helin, Tuule-Mai, and Loits, who were a lovely distraction from my work and studies.

Finally, my deepest thanks to my wife, Kristi, for all the support and patience and love. She was always motivating me to finish my thesis, and thankfully she never gave up. I am very lucky to have you by my side.

This research has been supported by European Regional Development Fund.

Abstract

Consolidation of Crowd-Sourced Geo-Tagged Data for Parameterized Travel Recommendations

The research covered in this thesis is focused on different aspects of the task of creating automated recommendations for tourism, focusing mostly on places of interest like beautiful views, architectural landmarks, charming areas etc.

A significant amount of work has been spent on designing and developing actual recommender systems - Sightsplanner, Sightsmap and the automated recommender of Visit Estonia - and their data harvesting methods in order to create a platform for showing the feasibility of the new methods designed and experimented with.

The main results of our research are split between three subfields:

- Knowledge engineering: we have shown how to formalize fuzzy and uncertain POI categories along with suitable ontologies and reasoner-based algorithms for object matching and score calculation in a real-life context of actual POI-s, available data and easily expressable user preferences.
- Machine learning: we have designed a learnable detection system for detecting duplicate POIs from different databases, usable for cross- category, cross-language and cross-city datasets.
- We show that learning on Tallinn eateries improved the algorithm parameters to such a degree that on Riga data containing also museums and galleries it gave us 98% accuracy versus 85% accuracy achieved by tuning the algorithm parameters manually.
- Knowledge extraction: we have designed an algorithm for high-quality keyword extraction from short crowd-sourced POI descriptions in different languages, able to find a suitable name and to add suitable types to the POI. Our clusterization algorithm is able to merge the POIs based on the extracted data: on the Panoramio and Wikipedia data about U.K. and French locations it was able to find 56% of Wikipedia objects from the textual titles/annotations of Panoramio pictures in the area.

Kokkuvõte
Kasutajatelt kogutud andmete integreerimine parametriseeritud reisisoovituste loomiseks

Käesolev doktoritöö käsitleb automaatse turismisoovitaja loomise erinevaid aspekte, fookusega visuaalselt atraktiivsetele ja huvitavatele kohtadele. Peamine rõhk töös on turismiobjektide andmete kogumisel ja töötlemisel.

Töö käigus on tegeletud erinevate soovitussüsteemide - Sightsplanner, Sightsmap ja soovitaja Visit Estonia veebilehel - disainimise ja arendamisega ning andmete kogumise, töötlemise ja integratsiooniga nende süsteemide jaoks.

Peamised teadustöö tulemused on järgmised:

- Töös on näidatud, kuidas rakendada tõenäosuslikku ja hägusloogikat turismiobjektidele sobivushinde arvutamisel, kasutades ebakindlaid kategooriaid, ontoloogiaid ning tõestajapõhiseid algoritme. Aluseks võetakse olemasolev turismiobjektide andmestik ning lihtsasti esitatavad kasutaja-eelistused.

- Rakendades masinõppemeetodeid töötatakse välja süsteem, mille abil tuvastatakse samu reaalseid objekte kirjeldavad kirjed erinevatest andmeallikatest. Loodud algoritmiga tuvastati Tallinna söögikohtade andmestiku pealt õppides Riia turismiobjektide, sh muuseumid ja kunstigaleriid jms, kattuvus täpsusega 98%. Eelnevalt oli see täpsus käsitsi seadistatud parameetritega 85%.

- Töös on loodud algoritm, millega tuvastatakse turismiobjektide nimi ja kategooria, tarvitades selleks kasutajate sisestatud kirjeldusi objektide kohta. Saadud andmete põhjal on võimalik integreeerida andmed erinevatest andmeallikatest. Kasutades Panoramio piltide allkirju suutis algoritm leida umbes 56% Suurbritannia ja Prantsusmaa Wikipedias esinenud turismiobjektidest.

Appendix 1

I

A. Luberg, J. Pindis, and T. Tammet. Sights, titles and tags: Mining a worldwide photo database for sightseeing. In *Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics*, WIMS 2020, page 149–158, New York, NY, USA, 2020. Association for Computing Machinery

Sights, titles and tags: mining a worldwide photo database for sightseeing

Ago Luberg
ago.luberg@taltech.ee
Tallinn University of Technology
Tallinn, Estonia

Jakob Pindis
jakob.pindis@taltech.ee
Tallinn University of Technology
Tallinn, Estonia

Tanel Tammet
tanel.tammet@taltech.ee
Tallinn University of Technology
Tallinn, Estonia

ABSTRACT

The paper focuses on calculating suitable place names and descriptive tags for large photo collections of visually interesting sights. The core dataset analyzed contains 45 million crowd-sourced geotagged pictures of the Panoramio database. We present several methods for analysis along with machine learning experiments for tag recommendation and suggest a manually built taxonomy of tag categories, based on the analysis of most widely used tag-like words in the photo titles, along with their popularities. The methods, selected tags and the taxonomy can be used for building different tourism applications for visually interesting sights.

CCS CONCEPTS

• **Information systems → Geographic information systems**; *Clustering and classification.*

KEYWORDS

crowd-sourced mapping, popularity analysis, photo tagging, POI categorization

ACM Reference Format:
Ago Luberg, Jakob Pindis, and Tanel Tammet. 2020. Sights, titles and tags: mining a worldwide photo database for sightseeing. In *WIMS '20: AThe International Conference on Web Intelligence, Mining and Semantics, June 30 – July 03, 2020, Biarritz, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3405962.3405987

1 INTRODUCTION

Sightsmap[1], see [11] for the initial report, is a world-wide tourism recommender with a focus on finding beautiful places from a heatmap of crowd-sourced touristic photos in the Panoramio system. The visual popularity data is augmented by extracting categories from multilingual photo titles and by merging and processing other crowd-sourced geotagged datasets, most importantly Wikipedia and Foursquare.

[1]http://sightsmap.com

WIMS 2020, June 30th - July 3rd, 2020, Casino Barrière, Biarritz - France

ACM ISBN 978-1-4503-7542-9/20/06…$15.00
https://doi.org/10.1145/3405962.3405987

In the current paper we focus on the issues of using the Panoramio photo titles for constructing suitable place names, selecting representative tags and building a practically useful taxonomy of tag categories.

The paper is organized as follows. After an overview of the related work we give a description of the Sightsmap system, which forms the basis for the current work on analysing and using the photo titles. In the next section we describe the algorithms and methods for title analysis, place name and tag calculation. We then present the results of our experiments of machine learning for tag selection and identify the methods which perform best on a relatively small set of labelled data. In the final section we present a suggested taxonomy of tag categories for practical use.

Some of the relevant calculated datasets are available at https://github.com/tammet/sightsmap.

2 RELATED WORK

The tagging / naming method we propose and employ uses a large number of photos to automatically tag popular touristic places. We do not use the actual contents of the pictures: instead we use the user-defined titles along with the location. A significant amount of work has been done in the area of image recognition in order to detect the objects in the pictures. A similarly popular area of research is to tag text (news, research papers etc.). Our case is more specific: we do not use picture contents and we do not have long texts to work with. We will give an overview of related work which covers ideas similar to ours.

In literature the task of finding tags is also called tag recommendation. In [3] the authors give a survey of different tag recommendation methods. Tagging is described as describing and organizing the content of the objects. In the survey they propose a taxonomy for tag recommendation. In our research, we are dealing with object-centered tagging. We are using context (photo titles) to provide tags to clusters of images (or to points of interests).

Authors in [19] and [20] use images from Flickr social network to indicate spatial areas of interest and events. They divide an area of predefined bounding box (a city center for example) into smaller groups of fixed size called geo clusters. Flickr photos have user defined tags which they use to identify the places of interests. By analyzing the textual data over time, they defined the importance of the tags and selected the most important ones to represent places and events.

A very detailed and thorough automatic picture analysis is presented in [16]. They have described the whole pipeline on how to cluster geotagged pictures, classify those and how to link clusters with Wikipedia. They use a subset of European cities and extract about 220 000 pictures for those places. For clustering they use both

the visual and text-based similarity. They point out that if they used only text for clustering, the result precision would be about 60% compared to 98% in case of visual clustering. After clustering, they label the clusters. For labelling, they use the frequent itemset mining algorithm which gives speed and scalability. In our case, we use statistical approach to find label candidates.

After they have found the label candidates for their clusters, web search is ran to detect Wikipedia articles. From every result they match the pictures of the article with the pictures of the cluster. If the pictures match, they connect the Wikipedia article with the cluster. Having a link to Wikipedia, they will have an annotation to the clusters which can be used in different ways, for example to get the name of the object.

The article is dealing with the very same topics we are interested in. They have described all the steps in the pipeline of labelling the picture clusters. We are not using visual comparison of pictures as proposed in their article, nevertheless the methodology mentioned is very interesting and as can be seen, there are similarities in our approaches. Some points which are worth looking into to improve our results: distinguishing between objects and events, looking into different textual similarity measures. In addition, we do not have clusters of pictures in the same sense they propose. In our case, the same picture could be used for several objects.

Assigning geographic coordinates to objects (e.g. pictures) is a popular topic. In [21] the authors present different ways to select terms which help assigning coordinates to Flickr photos and Wikipedia articles. One of the methods they mention is geographical spread (originally presented in [6]), which is used to find location-relevant terms. The idea is very similar to what we use in our method to find phrases which are candidates for tags. They look whether a term refers to a precise area (landmarks such as Eiffel Tower) or to a broader region (country of France). We use the similar idea, but do it country wise and aggregate the results over all the countries.

In [21] the goal of using the mentioned geographical spread is different. They want to locate tags which are location-specific. A picture with a tag which gives as much information about the location as possible helps to assign coordinates to the picture. Consider "Eiffel Tower": if the given tag is on the picture, it is relatively easy to assign coordinates to the picture. In contrast, if a picture has a tag "beach", assigning the exact location to the picture based on that tag is prone to errors.

Authors of [18] tackle a similar task - they want to assign tags to geotagged photos. Their approach is different: using existing tags from nearby pictures. For each photo they will look at other photos which are located close to the given photo and use the tags from those nearby photos. In addition they apply methods like visual similarity, different users employing the tag etc. to filter out suitable tags. In our methodology we take the near-by photos and use titles to extract tags. The idea to filter out some of the photos which may add noise is promising. In the work presented in our paper we have not applied filtering, but use a threshold to accept only phrases with a high enough appearance rate.

In [2] they propose a methodology to predict tags for images in Flickr. They use both visual and language components of images. The language part is user defined tags for the image. They suggest that as the user can tag images freely, the set of tags will be noisy. The paper proposes a mechanism how to normalize tag usage by providing related tags for an image. In their evaluation they find that language component method outperformed the visual and combined components.

In [14] the authors automatically build a database of geographic landmarks. They use Wikipedia and Panoramio as the data sources and employ web search to identify the names, categories, coordinates and ranks for geographical objects. For names they use Wikipedia articles and links to other pages. In addition to Wikipedia, they extract information from the Panoramio titles. Given a coordinate, they find pictures within the specified distance. Then they extract capitalized words which are adjacent to the geographical concept (e.g. Museum).

Similarly to our approach, they use the first sentence of Wikipedia to extract the type for the object. For the objects which do not have a Wikipedia article, they extract the type from the candidate name which matches with a geographical concept: this will be the first candidate for the object type. A web search for the object name is executed and the geographical terms are collected from the top results' web pages. Extracted terms will be type candidates for the given object. Given type candidates are compared with search queries combined with the object name. The candidate which gets the most of the search results is assigned as the type for the object.

In our approach we do not use a fixed list of geographical terms as type candidates. Instead we learn the possible candidates from the picture titles and use additional crowdsourced portals to extract possible types for the objects. Usually there are several different categories which apply for an object, therefore our approach finds a set of tags.

There is a lot of work done in the field of visual categorisation to detect type or the object in the picture or assign geographical location based on the contents of the picture. In [10] the authors use visual folksonomy to obtain tags for visually similar pictures. Pictures with user generated tags are extracted for image features. For non-tagged pictures the tags from similar images are used. In [9] the authors classify landmarks for large-scale image collection using visual, textual and temporal information. They show that if combined together, textual tag analysis improves the quality of other approaches. They use tags from Flickr to extend the feature vector for a picture. The authors show that tag-based textual models outperform models which use only image features. In our case we do not use photo tags: instead we make use of the photo titles, which add noise to the data.

3 OVERVIEW OF THE SIGHTSMAP SYSTEM

The goal of the sightsmap.com project is to build a world-wide tourism recommender along with a database of the sightseeing popularity and types of concrete places (POI-s) and wider areas in the world, using purely crowd-sourced data. The system has been viewed by over 1.5 million distinct users since its launch.

The main feature of the Sightsmap system is showing a zoomable and pannable touristic popularity heat map for any area in the world as an overlay on the standard Google maps (http://maps.google.com/). Popular areas on the map are labelled with an appropriate crowd-sourced name. Concrete popular places are also shown on the map

with colour-coded markers in the order of the relative popularity in the currently visible map area.

Due to the changes in the policy and technology of Google Maps API-s and the discontinuation of the Panoramio photo database, several of the important features of Sightsmap, like showing photos, are not available anymore. However, the main features are unchanged:

- The focus is on generating a zoomable worldwide heatmap of (visually) interesting places to visit.
- The main data source is Panoramio[2] with ca 40 million user-taken photos with focus on tourism and interesting/beautiful places; we harvest the locations and tags of photos and use these as a basis for location popularity heatmap and location tagging.
- The concrete POIs (places of interest) and their names are selected for popular areas, based on additionally harvesting and merging Wikipedia[3], Wikitravel[4] and Foursquare[5] along with mining and combining the popularities according to these sites.
- The popularity of places is calculated and used both for the selection of POIs to be displayed and for building a heatmap. By sightseeing popularity we mean the estimate of a number of people visiting the place and considering it as an interesting place for sightseeing, as opposed to popular places with no or very little potential for sightseeing, like hospitals, schools, gas stations, bus stops and airports.
- POIs are enriched with names and tags created by merging the names from the beforementioned sources and the multilingual photo titles/descriptions harvested from Panoramio.

Obviously, some of the abovementioned popular non- sightseeing places like schools and railroad stations may in some exceptional cases be sightseeing places as well: famous old colleges, Grand Central Terminal of New York, etc. Two separate important categories of objects in tourism industry - hotels and restaurants - are similarly ambivalent: on one hand, utilitarian and not necessarily a target or cause for travelling, on the other hand, a source of emotions and sometimes also an important motivation for travel.

Hence the project did not use data sources like TripAdvisor[6], Expedia[7], UrbanSpoon[8] or Zagat[9] which are primarily focused on specific categories, typically hotels and/or restaurants. Clearly, the hotels and restaurants are among the best crowd-described, -mapped, -reviewed and -rated tourism objects already.

The sightseeing popularity database we build is used by the sightsmap.com site for showing a zoomable and pannable touristic popularity heat map for any area in the world as an overlay on the standard Google maps. Popular areas on the map are labelled with an appropriate crowd-sourced name and colour-coded markers in the order of the relative popularity in the currently visible map area.

[2]http://panoramio.com of Google (discontinued by Google)
[3]http://wikipedia.org
[4]http://wikitravel.org
[5]http://foursquare.com
[6]TripAdvisor, see *http://www.tripadvisor.com/*
[7]Expedia, see *http://www.expedia.com/*
[8]UrbanSpoon, see *http://www.urbanspoon.com*
[9]Zagat, see *http://www.zagat.com*

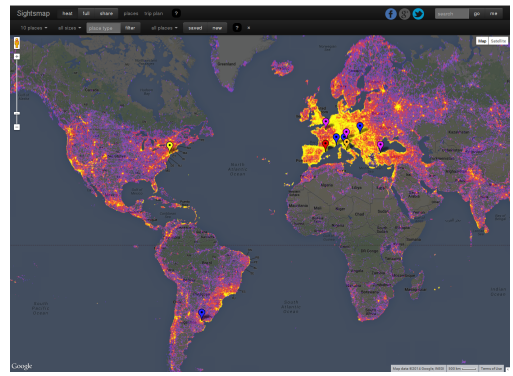

Figure 1: A screenshot of the heat map for most of the world on a single picture, with 10 top spots (1. New York, 2. Rome, 3. Barcelona, 4. Paris, 5. Istanbul) marked.

There are several advantages to using crowd sources is contrast to POI databases and guides created by experts in the tourism business. The crowd-sourced approach guarantees that there are no significant holes, i.e. interesting places and areas unmarked, and that the popularity estimates are relatively objective, which is hard to achieve by a small number of experts. Last not least, the popularity measurements can be done uniformly and comparably all over the world.

3.1 Different kinds of popularity and data sources

Although our methods focus on detecting sightseeing popularity, the notion itself is ambiguous and contains several different subcomponents: visual attractiveness, general public awareness about the place, the number of actual physical visitors etc. Each of the data sources used covers some components better than the others, effectively complementing each other. The data sources have been harvested using the public web API-s for Panoramio[10] and Foursquare[11] or downloaded in the already converted semantic format: Wikipedia[12] is downloaded in the form of the DBpedia RDF database, later complemented with the public Wikipedia logfiles. Harvesting and downloading has been performed during 2012 and 2013.

- Our main data source Panoramio.com represents the visual component of sightseeing: something beautiful or interesting to see. Panoramio contains ca 44 million geotagged photos uploaded by users. For several reasons, the Panoramio photos are dominated by the touristic and sightseeing interest, in contrast to more private photos on Flickr[13]. Google maps and Google earth[14] use the Panoramio photos as their photo

[10]https://www.panoramio.com/
[11]https://foursquare.com/
[12]http://en.wikipedia.org
[13]http://www.flickr.com/
[14]http://www.google.com/earth/

layer. We have downloaded only the metadata - location, photographer, title - not the actual photo files.

- The second data source Wikipedia represents the general public awareness about the place. We could safely say that all the interesting places, historic events, people etc. with public interest above a certain threshold do have a Wikipedia article. Places and historic events are normally geotagged in Wikipedia. The popularity - the exact number of readings in a selected time period - of each Wikipedia article can be obtained from the publicly available logfiles. We are using ca 700 000 geotagged Wikipedia articles with types which do not indicate noninterestingness for touristic purposes (like articles about plants, animals, people). We use full logfiles for two days, one selected from summer, the other from winter.
- The third data source Wikitravel (http://wikitravel.org/) essentially complements Wikipedia. Places above a certain touristic interestingness threshold normally have a Wikitravel article corresponding to some Wikipedia article. We are using the list of existing Wikitravel article names to detect whether a Wikipedia article has a complementing Wikitravel article as well.
- The fourth data source Foursquare gives an estimate of the number of people actually visiting the place. A large percentage of visits and a large percentage of Foursquare places are used and created by the local people visiting offices and eating lunch. Foursquare, differently from all the above sources, has a fairly detailed and well-used system for the crowdsourced typing of places. We have downloaded not the whole Foursquare places database, but only ca two million places, preferring the places Foursquare presents when asked for a circle around some of the top hotspots we have previously found out from the analysis of the Sightsmap photos. We harvest several concentric circles around each place previously determined to be visually popular enough: small circles for objects in the cities and large circles outside or around the cities. In the other words, we have only downloaded the more popular Foursquare places in the neighbourhood of the more visually popular (world-wide) places.

3.2 Heat map generation, labelling and data merging

The heat map generation has two separate outcomes. First, it generates the visual heat map overlays for the map. We use the browser-based Google maps as the underlying map. Second, it generates a detailed popularity data for each small rectangular area (a pixel on the heat map) for each zoom level, which is later used for labelling, harvesting additional information etc. The heat map generation is done separately for six different zoom levels of the world, each with each own granularity. Additionally, the seventh layer is a set of high-resolution heat maps, each typically covering one city, created for ca 15000 top spots in the world. The resolution of these high-resolution heat maps depends on the popularity rank of the hotspots: the more photos, the higher the resolution, up to the street level for the top 500.

Our algorithm takes into account both the number of photos and the number of separate photographers in the Panoramio database for each area. The colour of each pixel on the heat map is calculated by a logarithm-like root function, different for each zoom layer. We use one byte for the colour information, with the the top popular places being bright yellow, followed by orange, red, purple and blue hues.

3.3 Wikipedia labels

The pure visual popularity heat map lacks a clear indication of what exactly is there in a hot area. In short, the top spots in each view have to be marked and the markers should ideally contain the name and the pointers to the most relevant information about the places.

Our basic solution for creating these markers, finding the titles and providing pointers is to look for a most popular geotagged Wikipedia article at or very close to each top hotspot at each heat map grid. Articles with an obviously unsuitable type (like plants, animals, and people) are excluded. This method guarantees that, for example, on the whole-world view where each hot spot pixel corresponds to a relatively large area, we automatically get the Wikipedia city articles as the most popular, but as we zoom in, the area for each pixel becomes smaller and we will start getting markers and articles about villages, beaches, castles etc.

The actual algorithm is the following. First we cluster the heat map dots to avoid showing lots of markers very close to each other. Then we look for the most popular Wikipedia articles near the hotspots: the higher-ranked a heat map spot is, the larger the area to search. If nothing is found or the found article has a much lower popularity than the heat map spot, we do not attach anything to the hotspot. Otherwise we connect a hot spot to the Wikipedia article plus the corresponding Wikitravel article, if available.

In order to generate the popularity data and a popularity-sorted list of Wikipedia articles we use the Wikipedia logfiles plus an additional coefficient giving a significant bonus to Wikipedia articles with a type suitable for sightseeing, for example, world heritage sites.

It is worth noting that knowing a highest-ranked Wikipedia article for an area helps users to google for more, since the article always gives us a title of the place to look for.

3.4 Merging with Foursquare

The ultra-high-res heat maps for which we load Foursquare data is populated with the combined Wikipedia and Foursquare markers for top spots in the heat map, using an algorithm which - similarly to the Wikipedia labelling algorithm from the previous chapter - first tries to associate Wikipedia and Foursquare objects to the most popular places on the map and finally interleaves the remaining, unmatched top Wikipedia and Foursquare articles to the mix, even if they are not located near a visually attractive spot.

Foursquare places are merged with Wikipedia articles using an algorithm which takes into account both the geographical distance and a similarity of the name of the place with the name of the article. In order to be merged, both of these parameters must be sufficiently similar.

Foursquare locations are ordered based on the combination of different users ever checked in and the type of the place. First, we exclude both geotagged Wikipedia articles and Foursquare locations with obviously non-geographic or non-sightseeing type (homes,

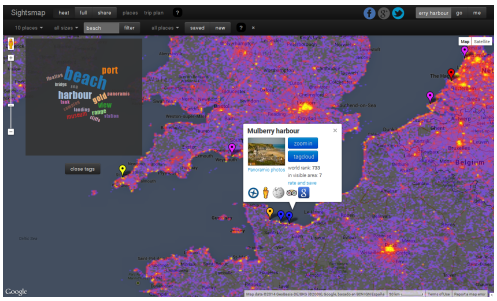

Figure 2: A screenshot of the selection by the "beach" tag in the English Channel area with a tag cloud for the Mulberry harbour area.

offices, bus stops etc.). Second, we add bonuses to articles and locations based on the suitability of their type: for example, castles, churches and public squares get different bonuses.

In most cases the geographical coordinates of the underlying visually popular spot, the closest popular Wikipedia article and the corresponding Foursquare location (close both by coordinates and the name), as well as the name of the article and the location are noticeably different. We use a relatively complex heuristic algorithm to determine the most suitable name and coordinate to present for the user as a marker. The percentage of errors our algorithm makes varies a lot for different zoom levels and regions and has not been measured with a sufficient quality to present it in the paper.

4 TAGGING OBJECTS

In order to enable searching objects and areas by type, we attach descriptive tags to several hundred thousand most popular objects. Tags are calculated from photo titles and in some cases augmented with a Foursquare type of a place.

The actual usage of tags in the sightsmap.com web application is twofold. First, a user can click and see a visual tagcloud of any important marker, giving an immediate rough idea of what are the interesting aspects of this particular place. Second, a user can start typing a word into the filter-by-type box, with autocompletion built in to dynamically present a list of all the matching tags, categories and their generalizations. Selecting a tag or a general category filters out all the locations where this tag or tags under the selected category have an importance over a certain threshold. We are satisfied with the usability of the described setup.

The following work is performed on the photo titles of ca 175 thousand most popular small areas (top spots) worldwide, as identified by the popularity analysis described in the previous section. Clearly, some of the top spots contain just one visually interesting sight, while some contain several close-by sights.

4.1 Multi-language titles

Before analyzing a picture title, we attach a country code to every picture, using shape files from http://thematicmapping.org/downloads/world_borders.php.

This dataset has only the main borders of the countries, plus bigger islands. For some pictures the surrounding country cannot be determined: in those cases we have found the closest border and assigned the picture to that country.

Knowing a country code for a picture gives an opportunity to distinguish languages. It also helps to process the picture titles in order to use country-based filters or properties. For example, consider a popular phrase present in a certain percentage of picture titles inside one country. This popular phrase might be something general (beach, museum) or the name of one specific object (Eiffel Tower). If the pictures with the given phrase are drawn on the map, we can see the coverage area for the given phrase in a country. In case the coverage area is relatively large - compared to the whole country area - then the particular phrase is probably a general tag and could be considered a type of the object. If the coverage area is rather small - for example, in case of Eiffel Tower only Paris is covered - we could say that this phrase matches a specific object: typically it is the name of the object.

Phrases which appear in different countries all over the world indicate a general tag for an object (museum, beach, church). For example, a word church is present in titles in at least 150 countries. Eiffel Tower appears a lot in France, but is not used much in other countries.

We use automatic translation for popular phrases for every language. Since the translation process for large quantities is expensive, we select about 500 to 1000 most popular phrases for a country. All of those phrases are then translated into English using Google Translate. After the translation process we merge phrases from different countries and from different languages. There are about one thousand different phrases which are popular in the photo titles all over the world.

4.2 Missing titles and types

For approximately seventy percent of the most popular areas we are able to find a local Wikipedia article or a Foursquare location providing both a likely name for the area and a likely type. In the Wikipedia case we use the first sentence from Wikipedia to extract type information. For example, the Wikipedia article about Stonehange has the following first sentence: "Stonehenge is a prehistoric monument in Wiltshire, England ...". The first sentence is often in a form "object is/was/are/were type" where we try to extract the type part. Another example for Tallinn Town Hall: "The Tallinn Town Hall (...) is a building in the Tallinn Old Town, Estonia ...". As can be seen from the last example, the indicated type can be very general.

Since about one third of our top spots is not connected to Wikipedia or Foursquare, extracting type or the title from concrete article is not available. In the next sections we will look into different methodologies we apply to our data to enrich it and present a novel idea which directs us towards getting the types and the title of a top spot.

4.3 Phrase extraction

Our approach is to use photo titles to understand what does the photo represent. Typically the title contains a name of the object or provides a hint about the object type, like "mountain" or "big church" etc. There are certainly titles with non-informative names

like "me in 2010", "this was so cool" etc. As the aim of Panoramio was to enrich Google Earth and Google Maps with a layer of visual aid for different areas of the world, the ratio of cat pictures are low. Therefore the majority of the pictures have meaningful titles.

For every top spot we select an area around it, with the size depending on the popularity of the spot. Every picture in this area is processed. The title of the picture is tokenised into lower case words. We ignore commas, full-stops etc. For every tokenised title we will find the n-grams for n from 1 to 4. An n-gram is combined by taking n consecutive words from the title. A simple example: given a title "A picture of Big Ben", we will end up with tokens: "a", "picture", "of", "big", "ben". All 1-grams are: "a", "picture", "of", "big", "ben". Followed by 2 and 3-grams: "a picture", "picture of", "of big", "big ben", "a picture of", "picture of big", "of big ben". And 4-grams are: "a picture of big", "picture of big ben".

All the pictures around the top spots are processed for n-grams. We then select the most frequent n-grams. If the phrase is present in at least 10 percent of the titles from all the pictures in the area, the phrase will be stored as a candidate. The candidates are then used to deduct titles and types for the objects.

4.4 Names versus descriptive tags

A photo about a concrete object will probably have a title with the object's name. People taking pictures of Eiffel Tower will usually mention it in their picture titles. Obviously, pictures which mention the Eiffel Tower are located mainly around the object itself. A phrase which occurs only on the pictures from a small area is assumed to indicate a name of the object.

A lot of pictures do not mention a name of the object, but indicate a general type or description of the object. If the user does not know the name of the object in the picture, she can still write a general description or type. A phrase (which could also be a part of the title, like "tower") which occurs in different locations around the country or world is likely to be a general descriptive tag. In order to find such tags, we look at how often is a certain phrase used and how wide an area does it cover - the wider the area, the more general / common the tag is. For example "beach" is used in very different places, so we can assume this is a general term which is used for several objects, therefore it is a descriptive tag.

Finding a phrase coverage for every country separately gives us more options to analyse the results and draw conclusions. As mentioned earlier, we have to tackle the language problem. For each country we can assume the language of origin of the title and translate the phrases into English. There is no need to translate every phrase we find: instead, we translate just the top 1000 phrases by occurrence. In addition to the occurrence we also check the coverage, i.e. how large a percentage of the country is covered by photos containing a concrete phrase. For example French word "plage" which translates to "beach" covers both western-northern and southern side of France. As the word is popular and covers wide area (compared to some local sightseeing), the word will be translated into English.

After we have found the top phrases for every country, the results are combined to form the worldwide set of tag words. Only phrases which are present in half of the countries are used as descriptive tags. The descriptive tags will be then propagated back to the top spots. Every top spot has a number of tag candidates: however, only those which ended up being tags in the whole world context will be considered.

5 EXPERIMENTAL RESULTS FOR PHOTO TITLE ANALYSIS

In this section we will present some results obtained using the process described earlier. We note that for our experiments we have only taken into account one-word tags. Phrases with two or more words are not considered.

We have about 45 million geo–tagged pictures from Panoramio. In addition to latitude and longitude, ca half of the pictures have a title - for us, those three fields are the main source of data. Additionally we have the owner identifier and more metadata along with the main fields.

As already mentioned, we have grouped pictures together by the location. Based on the amount of pictures in the group, we have selected 175.000 of the largest groups as our top spots. One top spot should point to a concrete tourism object: to be more exact, a visually attractive object based on the pictures taken. We use Wikipedia, Foursquare and also Geonames to match the top spots in order to get additional information about the location. The main goals are to get the name (what is shown on the pictures) and the types of objects on the pictures.

We will focus on the steps towards finding types for the objects. More concretely, we extract descriptive tags for every top spot: these are later used to filter out types. From 175.000 top spots we end up generating 28 million word n-grams or phrases. As we focus on one-word phrases, most of those generated n-grams are single words. On average, every top spot gets about 150 phrases attached.

For every country we find the area which a phrase covers. We use simple bounding box for all the phrase locations. If the bounding box covers more than 30% of the country's area and the phrase is present in at least 0.05% of the pictures of the given country, the phrase will be used as a country phrase. In total we have 90.000 country phrases. In our database we distinguish between 235 countries, which gives 380 phrases for a country on average.

A more indepth view of the process is described in Algorithm 1. The algorithm uses the following thresholds:

- $T_{freq} = 0.0005$ (0.05%) - a tag has to be present in that ratio of pictures in a country to be considered as a candidate;
- $T_{bbox} = 0.3$ (30%)- a bounding box of the pictures with the given tag has to cover the given ratio of the country's land area;
- $T_{country\_freq} = 0.3$ (30%) - the ratio in how many countries the tag has to be present (as a candidate, e.g. covers the prvious requirements) to be counted as a global tag.

5.1 Comparison of tag recommendation methods

To evaluate our tagging accuracy we have taken UK photos and compared the results with other phrase extraction methods. As our initial data is a set of titles or text in general, we use different language / text analytics tools. We have used UK as the most of the titles are in English and we will not use translation.

Data: Countries C, pictures P
Result: Popular global tags *tags*
tags = new list for popular tags
tag_countries = new hash map (tag → country objects)
for $c \in$ *countries* C **do**
  tag_pic = new hash map (tag → picture objects)
  for $p \in$*pictures for country* c P_c **do**
    for $t \in get\_words(p_{title})$ **do**
      if $t \notin tag\_pic$ **then**
        $tag\_pic[t]$ = new list
      end
      // add picture into the list by tag
      $tag\_pic[t].add(p)$
    end
  end
  popular_tag_pic = new hash map (tag → picture objects)
  for $t \in keys(tag\_pic)$ **do**
    // on how many pictures does the tag exist
    $tag\_freq = |tag\_pic[t]|/|P_c|$
    if $tag\_freq > T_{freq}$ **then**
      /* translate tag depending on the country's language to English */
      $tag\_tr = translate(t, c)$
      $popular\_tag\_pic[tag\_tr].addall(tag\_pic[t])$
    end
  end
  sorted_tag_pic =sort *popular_tag_pic* by picture count
  for $t \in keys(sorted\_tag\_pic)$ **do**
    // calculate bounding box
    $bbox\_area = get\_bbox\_area(sorted\_tag\_pic[t])$
    if $bbox\_area/C_{area} > T_{bbox}$ **then**
      /* this tag is frequent and covers wider area, will use this for the given country */
      $tag\_countries[t].add(c)$
    end
  end
end
for $t \in keys(tag\_countries)$ **do**
  /* in how many countries the tag was found popular enough */
  if $|tag\_countries[t]|/|C| > T_{country\_freq}$ **then**
    $tags.add(t)$
  end
end

Algorithm 1: Finding relevant tags

We have manually annotated 200 objects with different types and locations all over UK. We will use those to validate our results. In addition, we will be using 100 of those to train our own models to compare with pretrained models.

Table 1: The size of data we work with.

Panoramio pictures	45M
Top spots	175k
Top spots phrases	28M
Country phrases	90k
Countries	235
Top spots tags	2.8M
Different tags	944

We have taken 100 globally most popular phrases (stopwords and only small area phrases are filtered out) and use those as our set of available tags (or categories).

- The first is our n-gram phrase extraction which uses most frequent phrases in one picture cluster. Only the phrases which are in our available tags set are used.
- scikit-learn TF-IDF ([15], [5]) model. We have used scikit-learn framework [12] and provided Count Vectorizer and TfidfTransformer modules. We use a built-in model with the vector created from all the pictures and find top 10 phrases with the highest TF-IDF score from the titles of the cluster of pictures. For example, Big Ben as a point of interest with nearby pictures. Only the highest 3 which match the global popular phrases will be used-
- Wikipedia supported TF-IDF [17]. In comparison with the previous one, this model is trained on Wikipedia Summary Dataset.
- Trained scikit-learn TF-IDF model. We use 100 manually annotated objects for training and two different classification algorithms: decision tree (DecisionTreeClassifier) and neural network (MLPClassified - Multi-layer Perceptron classifier).
- TensorFlow Universal Sentence Encoder model [4]. We use TensorFlow [1] pretrained Universal Sentence Encoder version 4. All the cluster titles (concatenated titles of all the titles of pictures in the cluster) are transformed into 512 dimensional vectors. The available tags are transformed into vectors. A cosine similarity function is used to find the top 3 most similar categories.
- Explosion AI spaCy pretrained model [7]. We use a medium sized web code model. The model is based on Common Crawl and OntoNotes 5 [22] sources. The vectors are 300 dimensional. A cosine similarity function is used for similarity measure. As the category is one word and the title of the cluster is usually several words, spaCy will use average simliarity between the category and title words.
- Facebook fastText pretrained model [8]. The model is pretrained on Common Crawl and Wikipedia data. Vectors are 300 dimensional, a cosine similarity function is used to compare titles.
- Stanford GloVe pretrained model [13]. The sources for the model are Wikpiedia from 2014 and Gigaword 5. Vectors are 50 dimensional. The model contains 6 billion words and 400 000 unique words. In addition, a larger model with Common Crawl data was tried. The larger model contains 42 billion words and 1.9 billion unique words. The larger model did not give any improvements over the smaller dataset. As the

smaller dataset performed better, we used it in our experiments.

We used two evaluation methods to compare the results. The first is using f-score Figure (3) and the other is using a visual representation of the classification Figure (4). We use scikit-learn provided f-score calculation functionality. As we allow multiple tags (three) for one cluster, we are using f-score with a built-in micro-average calculation. Micro-average depends on true positive, false positive and false negative results. In addition, we have to define what happens if one of the comparable objects has fewer classes than the other. For example, the case where the tag recommender method only provides one or two, but our manually annotated object has three classes. The missing classes will set to empty. When comparing the results, the f-score will be low as empty does not match the annotated class. This could be changed depending on the goal of the classification. For example, if the classifier finds fewer tags, there could be no penalty.

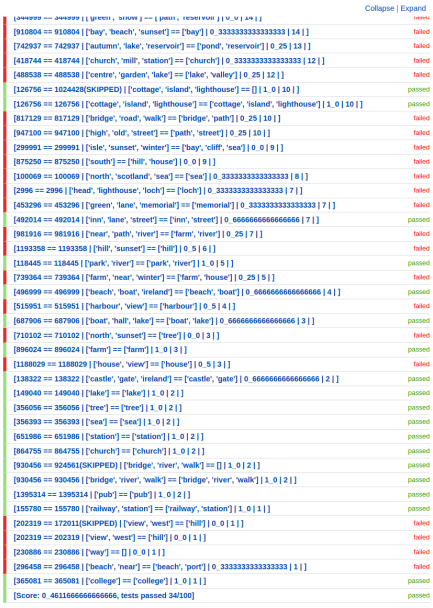

Figure 3: A screenshot of unit tests. Each row represents a test for one manually annotated cluster.

The results are shown in Figure 5. As can be seen, our trained models for TF-IDF did not perform well. The problem is the training data size: we only used 100 objects to train. Taken into account the small size, the result is not too far from other models. This is something we will look into in the future. Universal sentence encoder's low score can be explained with the lack of context in our input data. A picture title is usually not a whole sentence, but rather a fragment or a phrase. FastText is the highest pretrained model in our results. In the top three there are 2 TF-IDF based

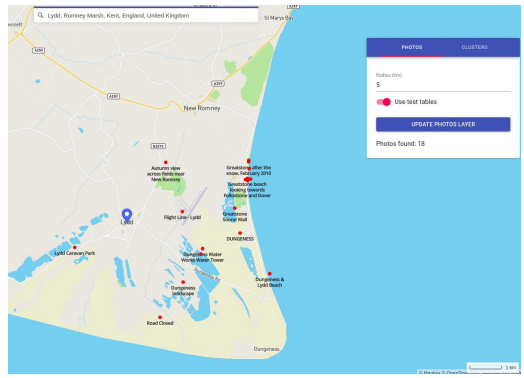

Figure 4: A screenshot of visual evaluation. The map can show each picture title individually, or the clusters and recommended tags. This way a larger set of clusters can be evaluated visually. See https://cs.ttu.ee/research/projects/sightsmap/.

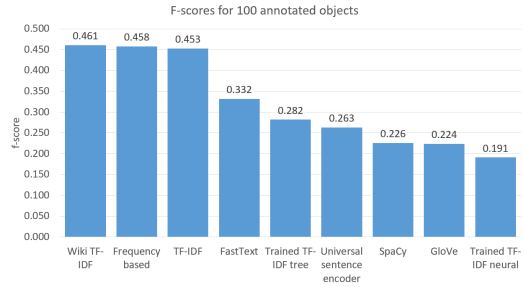

Figure 5: Results of all the different tagging methods.

models and our frequency based algorithm. We tried Wikipedia TF-IDF model with different configurations. Using more than 100,000 introduction articles, the results did not improve. We also tried to use only articles which contained phrases like: uk, United Kingdom, British isles, Britain. This did not improve the results either.

From the point of memory usage, fastText is the largest. GloVe, spaCy and Universal Sentence Encoder are average memory users, the rest use a rather small amount of memory. The fastest methods are frequency based and TF-IDF based methods. For pretrained models, it took some time to load the model into memory. It took about two hours to run the whole process of clustering and tagging for UK data of 1.4M pictures and one hour for merging pictures into clusters. SpaCy model took additional thirty minutes while the rest were faster.

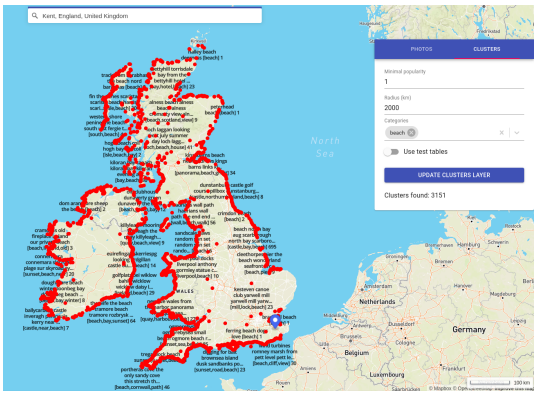

Figure 6: A screenshot of all the "beach" tags in UK based only on the image titles.

6 MANUAL SELECTION AND CATEGORIZATION

Despite the large amount of processed titles, the number of repeatedly used distinct tag words is not high at all. There are just approximately nine hundred tag words which are used more than twice. The top 10 of the list with the occurrence count is shown in the Table 2. Notice that some of the words like "saint" are likely to stem from place names which the algorithms failed to identify as names, while some words like "near" are clearly not suitable for use as tags.

Table 2: Most popular tags all over the world.

Tag	# of pictures	# of countries
view	1122018	197
beach	560349	138
church	545210	147
park	437798	135
bridge	377299	126
river	368806	135
lake	348466	117
road	339388	175
castle	330983	66
sunset	306240	176
street	292761	139
hotel	283653	166
old	281107	158
port	274308	93
saint	269436	52
tower	255302	94
station	250416	129

In figure 6 we have shown all the objects which have a "beach" tag based on the image titles. Each red dot represents a cluster of images, where the tag "beach" is in top 3 tags. Each such cluster is potentially a POI which is related to beach. The eastern coast is missing dots as the data we used to plot the map were cut of at the concrete longitude.

For practical use it makes sense to generalize the tag words under large categories. Obviously, some of the words like "near" or "sunset" are not really useful for categorization. All in all, we selected 179 suitable tag words from the top 900 of the full list. Clearly there is no sense in generating categories for which there are almost no instances. Since we used the word list without changes, it contains a few words with both a singular and plural version as well as some words like reka and kerk, for which the automatic translation did not find a proper English correspondence.

Analyzing the statistics of the tag words led us to create the following small list of actually useful categories, organized as a shallow taxonomy. The category names in the following tree start with the capital letters: all the other are tag words. We give a number of occurrences in the tag word and the number of summarized occurrences in the category after the word/category name. The number of tags shown in the tree are collected not from the whole Panoramio dataset, but from the 15.000 top spots: hence the numbers are ca ten times lower than in the table above. The dataset of these top spots along with the extracted tag candidates are available at https://github.com/tammet/sightsmap. The whole tree and its subtrees are sorted by the number of word occurrences.

- Landmark (103622) :
  - Historic (23626) :
    - Castle (5806): castle (4138), rampart (708), fortress (696), walls (264)
    - Monument/Memorial (5107) : monument (2266), cemetery (1215), memorial (1119), statue (221), grave (193), mausoleum (93)
    - old (7540), museum (2830), royal (553), ancient (440), historical (391), ruin (370), historic (329), abandoned (260)
  - Church (20125): church (9152), chapel (2337), cathedral (1814), temple (1614), holy (1170), monastery (1131), mosque (702), basilica (446), shrine (359), templom (351), cami (296), catholic (285), kerk (282), pagoda (186)
  - Harbour (9741) : port (3799), boat (2074), harbor (1776), ferry (861), ship (649), jetty (299), wharf (283)
  - Education (5001) : school (2853), university (1110), college (548), library (490)
  - Palace/Villa (3528) : villa (2030), palace (764), ranch (434), manor (224), villas (76)
  - house (7324), bridge (6730), station (4845), tower (4814), plaza (2087), market (1989), hall (1555), dam (1522), fountain (1500), farm (1457), lighthouse (1385), mill (1321), airport (1283), palace (764), quarries (723), tunnel (671), stadium (585), residence (321), windmill (240), hut (215), chalet (213), bazaar (57)
- Nature (78943) :
  - Sea (18636) : beach (4858), island (2906), sea (2233), bay (2186), coast (2097), marine (1460), lagoon (990), ocean (638), surf (366), sailing (223), reef (218), coastline (180), yacht (156), fjord (123), beaches (2)

  - Mountain (13186) : hill (3886), high (2347), mountain (2200), mount (1351), peak (1070), mountains (666), cliff (459), cliffs (323), volcano (310), hills (302), glacier (272)
  - Garden (6254) : garden (3350), gazebo (881), flower (780), flowers (778), patio (465)
  - Lake (4711) : lake (2240), pond (1407), reservoir (665), lakes (399)
  - River (4194) : river (2307), creek (1358), stream (393), reka (115), fleuve (21)
  - Valley (3444) : valley (2818), canyon (562), wadi (64)
  - Waterfall (2579) : waterfall (1321), fall (811), falls (447)
  - Wildlife (912) : elephant (201), pelican (147), wildlife (135), crocodile (85), buffalo (75), camel (73), iguana (69), lizard (53), leopard (25), hippo (18), impala (14), camels (12), elephants (4), hippos (1)
  - park (8246), tree (2524), rock (2306), trail (1916), forest (1888), green (1641), trees (842), nature (738), shore (677), grove (610), rocks (519), dunes (453), caves (422), desert (413), cave (383), waterfront (352), palms (226), jungle (222), dune (202), cactus (181), safari (117), mangrove (85), baobab (33), paddy (31)
- View (19291) : view (13886), panoramic (1937), landscape (1579), beautiful (1394), skyline (299), panorama (196)
- Accommodation (7090) : hotel (6950), guesthouse (89), housing (51)
- Relaxation (6067) : club (1905), pool (1721), resort (1359), spa (788), casino (294)
- Food and drink (5378) : restaurant (1993), inn (1334), bar (1214), cafe (563), pub (274)
- Art (3059) : statues (1029), sculpture (563), theater (556), gallery (298), theatre (226), art (200), culture (187)

7 SUMMARY

We have proposed several methods and algorithms for analysing photo titles of worldwide popular places, with the goal of finding both suitable place names and tags. We observe that our experiments with fully automated tag recommendation using machine learning did not give high quality results, likely due to a relatively small number of annotated top spots used in these experiments. Nevertheless, we identify machine learning methods which performed clearly better than others, thus giving a direction for future experiments. Finally we present a suggested manual taxonomy of selected tag categories, based on the analysis of most widely used tag-like words in the photo titles, along with their popularities. The methods, selected tags and the taxonomy can be used as a statistically grounded basis for building different tourism applications focusing on visually interesting sights.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Ashton Anderson, K Ranghunathan, and Adam Vogel. 2008. Tagez: Flickr tag recommendation. *Association for the Advancement of Artificial Intelligence* 3 (2008).

[3] Fabiano M Belém, Jussara M Almeida, and Marcos A Gonçalves. 2017. A survey on tag recommendation methods. *Journal of the Association for Information Science and Technology* 68, 4 (2017), 830–844.

[4] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, and others. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175* (2018).

[5] Hans Christian, Mikhael Pramodana Agus, and Derwin Suhartono. 2016. Single Document Automatic Text Summarization using Term Frequency-Inverse Document Frequency (TF-IDF). *ComTech: Computer, Mathematics and Engineering Applications* 7, 4 (2016), 285–294.

[6] Claudia Hauff and Geert-Jan Houben. 2011. WISTUD at MediaEval 2011: Placing Task.. In *MediaEval*.

[7] Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. (2017). To appear.

[8] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of Tricks for Efficient Text Classification. *arXiv preprint arXiv:1607.01759* (2016).

[9] Yunpeng Li, David J Crandall, and Daniel P Huttenlocher. 2009. Landmark classification in large-scale image collections. In *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 1957–1964.

[10] Stefanie Lindstaedt, Roland Mörzinger, Robert Sorschag, Viktoria Pammer, and Georg Thallinger. 2009. Automatic image annotation using visual content and folksonomies. *Multimedia Tools and Applications* 42, 1 (2009), 97–113.

[11] Ago Luberg, Priit Järv, and Tanel Tammet. 2013. Sightsmap: crowd-sourced popularity of the world places. In *Information and Communication Technologies in Tourism 2013*. Springer.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[13] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162

[14] Adrian Popescu, Gregory Grefenstette, and Pierre Alain Moëllic. 2008. Gazetiki: automatic creation of a geographical gazetteer. In *Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries*. ACM, 85–93.

[15] Shahzad Qaiser and Ramsha Ali. 2018. Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents. *International Journal of Computer Applications* 181 (2018), 25–29.

[16] Till Quack, Bastian Leibe, and Luc Van Gool. 2008. World-scale mining of objects and events from community photo collections. In *Proceedings of the 2008 international conference on Content-based image and video retrieval*. ACM, 47–56.

[17] Thijs Scheepers. 2017. *Improving the Compositionality of Word Embeddings*. Master's thesis. Universiteit van Amsterdam, Science Park 904, Amsterdam, Netherlands.

[18] Ana Silva and Bruno Martins. 2011. Tag Recommendation for Georeferenced Photos. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks (LBSN '11)*. ACM, New York, NY, USA, 57–64. DOI: http://dx.doi.org/10.1145/2063212.2063229

[19] E. Spyrou and P. Mylonas. 2011. Placing User-Generated Photo Metadata on a Map. In *2011 Sixth International Workshop on Semantic Media Adaptation and Personalization*. 79–84.

[20] Evaggelos Spyrou and Phivos Mylonas. 2016. Analyzing Flickr metadata to extract location-based information and semantically organize its photo content. *Neurocomputing* 172 (2016), 114 – 133. DOI: http://dx.doi.org/https://doi.org/10.1016/j.neucom.2014.12.104

[21] Olivier Van Laere, Jonathan Quinn, Steven Schockaert, and Bart Dhoedt. 2014. Spatially Aware Term Selection for Geotagging. *IEEE Transactions on Knowledge and Data Engineering* 26, 1 (Jan. 2014), 221–234. DOI: http://dx.doi.org/10.1109/TKDE.2013.42

[22] Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, Mohammed El-Bachouti, Robert Belvin, and Ann Houston. 2013. OntoNotes Release 5.0. *LDC2013T19, Philadelphia, Penn.: Linguistic Data Consortium* (2013).

Appendix 2

II

T. Tammet, A. Luberg, and P. Järv. Sightsmap: Crowd-sourced popularity of the world places. In L. Cantoni and Z. P. Xiang, editors, *Information and Communication Technologies in Tourism 2013*, pages 314–325, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg

Sightsmap: crowd-sourced popularity of the world places

Tanel Tammet[a,b], Ago Luberg[a,b], Priit Järv[a,b]

[a] Eliko Competence Centre
Tallinn, Estonia
ago.luberg@eliko.ee

[b] Tallinn University of Technology
Tallinn, Estonia
tanel.tammet@ttu.ee
priit@cc.ttu.ee

Abstract

We analyse and combine a number of world-wide crowd-sourced geotagged databases with the goal to locate, describe and rate potential tourism targets in any area in the world. In particular, we address the problem of finding representative names and top POIs for popular areas, with the main focus on sightseeing. The results are demonstrated on the sightsmap.com site presenting a zoomable and pannable tourism popularity heat map along with popularity-sorted POI markers for concrete objects.

Keywords: crowd-sourced mapping; popularity analysis; heat map; entity disambiguation

1 Introduction

The goal of this work is to build a world-wide database of the sightseeing popularity of concrete places (POI-s) and wider areas in the world, using purely crowd-sourced data. By sightseeing popularity we mean the estimate of number of people visiting the place and considering it as an interesting place for sightseeing, as opposed to very popular places with no or very little potential for sightseeing, like hospitals, schools, gas stations, bus stops and airports.

Obviously, some of the abovementioned popular non-sightseeing places like schools and railroad stations may in some exceptional cases be sightseeing places as well: famous old colleges, Grand Central Terminal of New York, etc. Two separate extremely important categories of objects in tourism industry – hotels and restaurants – are ambivalent as well: on one hand, utilitarian and not necessarily a target or cause for travelling, on the other hand, an important source of emotions and sometimes also an important partial motivation for travel.

As said, our work is focused on popular sightseeing places regardless of their category. Hence we are not using any data sources like TripAdvisor (http://www.tripadvisor.com/), Expedia (http://www.expedia.com/), UrbanSpoon

(http://www.urbanspoon.com) or Zagat (http://www.zagat.com) which are primarily focused on specific categories, typically hotels and/or restaurants. Clearly, the hotels and restaurants are among the best crowd-described, -mapped, -reviewed and -rated tourism objects already.

Fig 1. A screenshot of the heat map for most of the world on a single picture, with 10 top spots (1. New York, 2. Rome, 3. Barcelona, 4. Paris, 5. Istanbul) marked. Europe, especially the belt from Netherlands to Italy as well as the mountainous areas and the Spanish coastal areas dominate. In U.S. the mountainous areas in Utah and Colorado are well marked, in addition to coastal cities. The original picture is colour-coded as a proper heat map.

The sightseeing popularity database we build is used in the sightsmap.com site for showing a zoomable and pannable touristic popularity heat map for any area in the world as an overlay on the standard Google maps (http://maps.google.com/). Popular areas on the map will be labelled with an appropriate crowd-sourced name. Concrete popular places will be also shown on the map with colour-coded markers in the order of the relative popularity in the currently visible map area.

There are numerous application possibilities for such a database. First, it is already used for showing map overlays geared towards finding interesting POI-s to visit in any region, large or small, in a uniform manner anywhere in the world. Second, the database can be used as an input for a tourism recommender like Sightsplanner (Luberg et al., 2011; Luberg et al., 2012). Third, the database can be used for doing popularity analyses for the tourism industry.

There are also several advantages to using crowd sources as contrasted to POI databases and guides already created by experts in the tourism business. The crowd-sourced approach guarantees that there are no significant holes, i.e. interesting places and areas unmarked, and that the popularity estimates are, despite inevitable

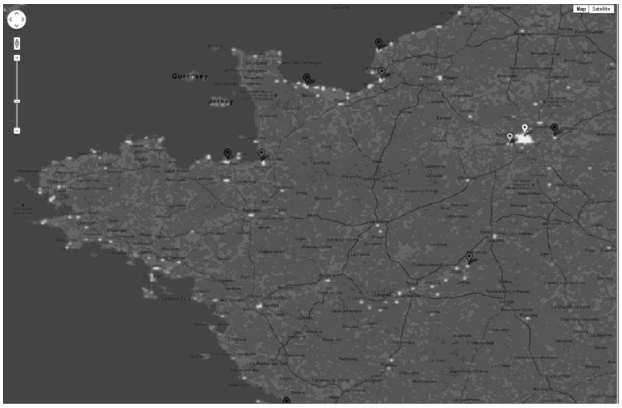

Fig 2. A screenshot of the heat map for the north-western France, with 10 top spots (1. Paris. 2. Versailles, 3. Euro Disneyland, 4. Mont Saint Michel, 5. Honfleur) marked. The castles of the Loire Valley form the central belt. The original picture is color-coded as a proper heat map.

Fig 3. A screenshot of the heat map for Manhattan, with 10 top spots marked. The open marker popup window links to the Wikipedia and Foursquare pages of the Metropolitan Museum of Art. The original picture is color-coded as a proper heat map.

fluctuations, relatively objective, which is very hard to achieve by a small number of experts. Last not least, the popularity measurements can be done uniformly and comparably all over the world.

In the next section we will provide a brief overview of the data sources and the main algorithms employed in our system. In the section 3 we will describe the relations between the data sources and the aspects of merging and enriching data in more detail. Section 4 will present experimental results and we will end our paper with related work and conclusion.

2 Different kinds of popularity and data sources

Although our methods focus on detecting sightseeing popularity, the notion is ambiguous and contains several different subcomponents (visual beauty/interestingness, general public awareness about the place, the number of actual physical visitors etc.). Each of the data sources used covers some components much better than the others; hence they complement each other well. The data sources have been harvested using their public web API-s (Panoramio (www.panoramio.com/) and Foursquare (https://foursquare.com/)) or downloaded in the already converted semantic format (Wikipedia (http://en.wikipedia.org) downloaded in the form of DBpedia RDF database, later complemented with the public Wikipedia logfiles). Harvesting and downloading has been performed during 2012.

- Our main data source Panoramio.com represents the *visual component* of sightseeing: something beautiful or interesting to see. Panoramio contains ca 44 million geotagged photos uploaded by users. For several reasons, the Panoramio photos are dominated by these with touristic and sightseeing interest (in contrast to more private photos on Flickr (http://www.flickr.com/)). Google maps and Google earth (http://www.google.com/earth/) use the Panoramio photos as their photo layer. We have downloaded only the metadata (location, photographer, title), not the actual photo files.

- The second data source Wikipedia represents the *general public awareness* about the place. We could safely say that all the interesting places, historic events, people etc. with public interest above a certain threshold do have a Wikipedia article. Places and historic events are normally geotagged in Wikipedia. The popularity – the exact number of readings in a selected time period – of each Wikipedia article can be obtained from the publicly available logfiles. We are using ca 700 000 geotagged Wikipedia articles with types which do not indicate noninterestingness for touristic purposes (like articles about plants, animals, people). We use full logfiles for two days, one selected from summer, the other from winter.

- The third data source Wikitravel (http://wikitravel.org/) essentially complements Wikipedia: places: above a certain touristic interestingness

threshold normally have a Wikitravel article corresponding to some Wikipedia article. We are using the list of existing Wikitravel article names to detect whether a Wikipedia article has a complementing Wikitravel article as well.

- The fourth data source Foursquare gives an estimate of the *number of people actually visiting the place*. A large percentage of visits (and a large percentage of Foursquare places) are done and created by local people visiting offices and eating lunch. Foursquare, differently from all the above sources, has a fairly detailed and well-used system for the crowd-sourced typing of places. We have downloaded not the whole Foursquare places database, but only ca 2 000 000 places, taking the places Foursquare presents when asked for a circle around some of the top hotspots we have previously found out from the analysis of the Sightsmap photos. We harvest several concentric circles around each place previously determined to be visually popular enough: small circles for objects in the cities and large circles outside or around the cities. In the other words, we have only downloaded the more popular Foursquare places in the neighbourhood of the more visually popular (world-wide) places.

3 Heat map generation, basic labelling and data merging

The heat map generation has two separate outcomes. First, it generates the visual heat map overlays for the map. We use the browser-based Google maps as the underlying map. Second, it generates a detailed popularity data for each small rectangular area (a pixel on the heat map) for each zoom level, which is later used for labelling, harvesting additional information etc.

The heat map generation is done separately for six different zoom levels of the world, each with each own granularity. Additionally, the seventh layer is a set of high-resolution heat maps, each typically covering one city, created for ca 15000 top spots in the world. The resolution of these high-resolution heat maps depends on the popularity rank of the hotspots: the more photos, the higher the resolution, up to the street level for the top 500.

Our algorithm takes into account both the number of photos and the number of separate photographers in the Panoramio database for each area. The colour of each pixel on the heat map is calculated by a logarithm-like root function, different for each zoom layer. We use one byte for the colour information, with the the top popular places being bright yellow, followed by orange, red, purple and blue hues.

3.1 Basic labelling with Wikipedia

The pure visual popularity heat map lacks a clear indication of what exactly is there in a hot area. In short, the top spots in each view have to be marked and the markers

should ideally contain the name and the pointers to the most relevant information about the places.

Our basic solution for creating these markers, finding the titles and providing pointers is to look for a most popular geotagged Wikipedia article at or very close to each top hotspot at each heat map grid. Articles with an obviously unsuitable type (like plants, animals, and people) are excluded. This method guarantees that, for example, on the whole-world view where each hotspot pixel corresponds to a relatively large area, we automatically get the Wikipedia city articles as the most popular, but as we zoom in, the area for each pixel becomes smaller and we will start getting markers and articles about villages, beaches, castles etc.

The actual algorithm is the following. First we cluster the heat map dots to avoid showing lots of markers very close to each other. Then we look for the most popular Wikipedia articles near the hotspots: the higher-ranked a heat map spot is, the larger the area to search. If nothing is found or the found article has a much lower popularity than the heat map spot, we do not attach anything to the hotspot. Otherwise we connect a hotspot to the Wikipedia article plus the corresponding Wikitravel article, if available.

As mentioned before, in order to generate the popularity data and a popularity-sorted list of Wikipedia articles we use the logfiles mentioned before plus an additional coefficient giving a significant bonus to Wikipedia articles with a type suitable for sightseeing, for example, world heritage sites.

It is worth noting that knowing a highest-ranked Wikipedia article for an area helps users to google for more, since the article always gives us a title of the place to look for.

3.2 Basic merging with Foursquare

The ultra-high-res heat maps for which we do load Foursquare data is populated with the combined Wikipedia and Foursquare markers for top spots in the heat map, using an algorithm which – similarly to the Wikipedia labelling algorithm from the previous chapter – first tries to associate Wikipedia and Foursquare objects to the most popular places on the map and finally interleaves the remaining, unmatched top Wikipedia and Foursquare articles to the mix, even if they are not located near a visually attractive spot.

Foursquare places merging with Wikipedia articles is performed using an algorithm which takes into account both the geographical distance and a similarity of the names of the place vs. the article. In order to be merged, both of these parameters must be sufficiently similar.

Foursquare locations are ordered based on the combination of different users ever checked in and the type of the place. First, we exclude both geotagged Wikipedia articles and Foursquare locations with obviously non-geographic or non-sightseeing

type (homes, offices, bus stops etc.). Second, we add bonuses to articles and locations based on the suitability of their type: for example, castles, churches and public squares get different bonuses.

In most cases the geographical coordinates of the underlying visually popular spot, the closest popular Wikipedia article and the corresponding Foursquare location (close both by coordinates and the name), as well as the name of the article/location are noticeably different. We use a relatively complex heuristic algorithm to determine the most suitable name and coordinate to present for the user as a marker. The percentage of errors our algorithm makes varies a lot for different zoom levels and regions and has not been measured with a sufficient quality to present it in the paper.

4 Labelling areas and merging objects: issues and improvements

The general idea behind labelling visual hotspots was briefly described above. Here we will present some main problems we have encountered and propose ways to improve our system.

For every visual hotspot we try to to find a matching Wikipedia article. A significant percentage of popular hotspots will get a match from Wikipedia. We try to find the name for non-matching objects by looking at Panoramio pictures nearby. We take a certain area around the hotspot (for example, 1 km radius) and look at the titles of pictures within that area. Based on this information we try to get the name of the object in the hotspot.

Table 1. An example of candidate list for pictures near Cliffs of Moher. The best match is has rank 1 and *n* 3 (marked with italics). Some less frequent candidates are omitted.

Candidate	n	Rank	Pos	Total	%
moher	1	1	656	859	76.4
of	1	2	631	859	73.5
cliffs	1	3	587	859	68.3
of moher	2	1	595	859	69.3
cliffs of	2	2	559	859	65.1
moher ireland	2	3	67	859	7.8
cliffs of moher	*3*	*1*	*534*	*859*	*62.2*
of moher ireland	3	2	64	859	7.5
cliffs of moher ireland	4	1	60	859	7.0

The title of the picture is tokenised into lower case words. We ignore commas, full-stops etc. For every tokenised title we will find the word *n-grams* for *n* being from 1 to 4. An *n-gram* is combined by taking *n* consecutive words from the title. A simple example: given a title "A picture of Big Ben", we will end up with tokens: "a", "picture", "of", "big", "ben". All *1-grams* are: "a", "picture", "of", "big", "ben". And all *4-grams* are: "a picture of big", "picture of big ben".

After finding *n-grams* for every picture in the area of interest, we take the 5 most frequent *n-grams* for every *n*. We will end up having up to 20 *n-grams* (5 most frequent for every *n=1..4)* for a hotspot which we consider name candidates.

An example candidate list for "Cliffs of Moher" (pictures near Lahinch, Galway in Ireland) is presented in Table 1. The column *n* stands for *n* used in *n-gram* (how many tokens is used to form up a candidate), *Rank* stands for rank in current *n* (1 being the most frequent *n-gram*), *Pos* ("positive" pictures) is a number of pictures which contain the given *n-gram*, *Total* represents the total number of pictures near by and *%* shows the percentage of "positive" pictures. We have marked the correct candidate in the table.

The given example illustrates already some problems we have with this methodology. After generating a list of candidates, we have to pick the correct candidate. Finding the correct one is not so straightforward. It is obvious that we cannot use the most frequent candidate as the final name, because it may-be just part of our final name. If our final name consists of 3 words, then every word alone in this name has at least the same or even higher frequency. This is very clear in the example: “cliffs”, “of” and “moher” all have higher frequency than “cliffs of moher” together.

The idea we have with the candidate selection is to find the longest candidate which has frequency above a certain threshold. For example, if the threshold is 30%, then we would find “cliffs of moher” to be the best candidate. To improve the precision, we are planning to apply machine learning to find the best threshold (or may-be even have additional indicators for the best pick in addition to frequency and term count).

Another problem is more related to the concept of taking pictures. It often happens that bigger (high) objects can be captured only from distance. It is very hard to take a picture of Eiffel Tower when being right in front of it. The same applies for our example “Cliffs of Moher”. The candidate list we presented earlier is actually taken from about 2 kilometres from the object itself (object location based on Wikipedia). Wikipedia location for the cliffs has about 400 pictures and 267 mention “Cliffs of Moher”, while 2 kilometres away the count of pictures is about 800 and 534 of those mention the correct object.

For our system, we actually need both those places. If later we want to have a recommendation of the best sightseeing places, we can prefer the distant location to take pictures. The 2 kilometre gap between the objects makes it harder to merge them into one. Currently we will have two separate objects (even though the name of two places could be the same).

In the next section we will present some experiments with Panoramio picture titles. All the work presented is based on the methodology described in the current section.

5 Experiments and Results

We use two different datasets for our tests: pictures from United Kingdom and pictures from France. For every popular place we have found up to 20 possible candidate titles. In order to evaluate our simple approach, we use Wikipedia to extract titles of popular objects. For every popular object we find a Wikipedia article with the same or close geocoordinates. In case there are several Wikipedia pages for one location, we try to take the most appropriate (popular and type-wise suitable). Obviously, not all visually popular locations have a Wikipedia entry. In our evaluation we only consider those locations which have a linked Wikipedia article. After generating all the *n-gram* candidates for a location we will see whether the Wikipedia name is within those candidates. Statistics about the datasets can be found in Table 2.

Table 2. Statistics about the datasets for UK and France.

Property	**UK**	**France**
Hotspots	14 768	13 621
Wikipedia objects	9753	9931
Panoramio picture count	1.4M	1.5M
Wikipedia object match	5458	5531
Match %	56%	56%

As shown in the Table 2, we were able to find about 56% Wikipedia objects from the Panoramio pictures. This means that the Wikipedia name matches (we allowed *Levenshtein distance* (Levenshtein distance, edit distance, http://en.wikipedia.org/wiki/Levenshtein_distance) up to 3) with one candidate. We outline several reasons why some objects are not found/matched:

- The number of pictures in the close vicinity is very low (or even zero). If we have an object and only 3 pictures mention that object, we want to look at pictures from the bigger area. We can extend the search area, and end up with 20 new pictures, but none of those mention the object we were looking for (all the new pictures mention some other object).

- Wikipedia and Panoramio coordinates do not match. We look only those matches which are close to each other. For our matching evaluation we need Wikipedia and Panoramio pictures to be very close. It may happen that the source data has somewhat rounded coordinates (0.01 difference in latitude or longitude number can mean 1 km distance). Another possibility is that some objects are usually pictured from a distance. A good example was given in the previous section about Cliffs of Moher.

- Different name variants. In Wikipedia, some objects have additional information like county or country in their titles. For the Wikipedia place

"Lincoln, England" we have found an n-gram "Lincoln", which is a correct match. These kinds of matches are not counted in our "match" number.

- The Panoramio title is too general. For some objects, there are a lot of pictures which indicate the name of the city or county where the object is located. For example, the case where there are 100 pictures near a certain Wikipedia object and only 3 mention the object itself. Other pictures mention the city, the county etc. It can easily happen that more general n-grams push the correct object out.

Our dataset for the described experiments has about 14 000 "hotspot" objects and about 10 000 Wikipedia objects. For the objects with Wikipedia articles, we could combine Panoramio and Wikipedia data to validate the title of the object. For the rest, we have to rely on Panoramio pictures (or on some additional external data source). Usually the title generated from the Panoramio title *n-grams* is not wrong, but it might be too general or a slightly different variation than Wikipedia article would have. We estimate that the Panoramio based object titles are correct in at least 56% of cases. If we add different name variations and more general objects, we might end up with 70-80 %.

6 Related Work

Heatmaps are used in various domains in order to visualise intensity of a certain values. We mention few which are also related to tourism. Fisher (2007) uses tile download statistics from Microsoft map server to present popular areas. He calls the system Hotmap. Every time a user looks a map, she downloads visible tiles from the server. Objects (and tiles) which are watched more often, have higher download numbers and they will become more popular for Hotmap. They present different ways to use heatmaps mentioning also a possibility to draw users' attention to prominent objects.

Kurata (2012) presents a potential-of-interest map based on Flickr pictures in Yokohoma. He present an interesting approach for finding popularity of objects where only pictures from non-local users are taken into account. Users who live in the city, are considered as non-tourists and their pictures do not add popularity. In our case, to find the name and the type of the object, we have to use pictures from local people. And it may happen, that those are even more accurate than tourist pictures, as a tourist may not know the exact name of the object. Kurata presents user evaluation which is very valuable and something we still have to organise for our recommender system.

Crandall et al. (2009) describe their system which uses image textual and visual features to group pictures into popular objects. They find a name and a descriptive picture for every popular object. Processing image textual information is very close to what we have presented in our paper. They use distinctiveness to order name candidates instead of using candidate name ratio to all pictures near-by the object. They present a machine learning technique usage for solving the problem of naming

the objects (where the photo is taken). Although they present that combining textual and visual features yield the best results, we keep our focus on using only textual information.

Alves et al. (2009) present KUSCO system which deals with enriching POI data. They extract information from search engine to gather web pages about a certain POI. Then they use natural language processing to extract concepts for objects. An interesting idea is to use WordNet (http://wordnet.princeton.edu/) concepts matched with words from the web pages. We have started working on something similar: we try to extract words from Panoramio picture titles and find similarity or distance between found words and WordNet concepts. We only consider certain concepts from WordNet which represent categories of POI: museum, restaurant, hotel, church etc.

Popescu et al. (2008) present a system which integrates Wikipedia and Panoramio in order to identify geographical names, categorise objects, find geographical coordinates and rank objects. They use Panoramio picture count as one possible rank for objects (more pictures means higher rank). They also try to find categories for objects where they use language processing from the first sentence of Wikipedia and web search. They compare their system with Geonames (http://www.geonames.org/), but they do not use Geonames as a source for their data. A lot of our ideas align with their proposed solutions: using Panoramio for ranking objects, merge objects with Wikipedia, try detecting categories from web search (something we are currently working on).

Popescu et al. (2009) present a multilingual geographical gazetteer creation based on Flickr, Panoramio, Wikipedia and web search. They detect place names using a vocabulary with geographical concepts. They also present object ranking and categorisation. They have improved some of the methods compared to their paper from 2008. They use Flickr instead of Panoramio. They have also published their gazetteer which can be downloaded (http://georama-project.labs.exalead.com/gazetiki.htm). We could evaluate our system against this dataset. However, we need to implement some additional functionality before doing the evaluation, in order to perform full range comparison.

Zheng et al. (2009) describe a system for building a world-wide landmark database. They use pictures from Picasa (http://picasa.google.com) and Panoramio along with Google Image Search (http://images.google.com) to download picture files. They also use textual information from Wikitravel to complement objects which are not present in pictures. They use picture and Wikitravel text information to find the name for the popular object. Image processing helps to detect different pictures about the same object which can be clustered into one group. In addition to image processing they use picture title word n-grams – the most frequent n-gram is used as the title for the group.

7 Conclusions and Future Work

We have presented the Sightsmap system with a goal to build a world-wide database of the sightseeing popularity of concrete POI-s. We are using purely crowd-sourced data: Panoramio, Wikipedia, Wikitravel, Foursquare. While the main goal is to detect popularity, first we have to tackle different data extraction and integration problems. We have presented experiments on finding an object name from the Panoramio picture titles. We have also described the way to gather information and to use different sources to calculate popularity for objects in the world. We have presented a heat map solution sightsmap.com, where all our data is put to use.

One of the future plans is to be able to recommend objects all over the world. The recommendation should be based on the interests of the tourist, hence we need to find a category for every object in the world. We have already started working on this goal and have briefly mentioned our ideas on the subject.

References

Alves, A., Pereira, F., Biderman, A. & Ratti, C. (2009). Place Enrichment by Mining the Web. In M. Tscheligi, B. de Ruyter et al. (Eds.), *Ambient Intelligence*, 5859: 66-77. Springer Berlin.

Crandall, D. J., Backstrom, L., Huttenlocher, D., & Kleinberg, J. (2009). Mapping the world's photos. In *Proceedings of the 18th international conference on World wide web:* 761-770. ACM.

Fisher, D. (2007). Hotmap: Looking at Geographic Attention. *IEEE Transactions on Visualization and Computer Graphics 13*(6): 1184-1191.

Kurata, Y. (20120) Potential-of-Interest Maps for Mobile Tourist Information Services. *Information and Communication Technologies in Tourist 2012:* 239-248. Springer Vienna.

Luberg, A., Järv, P., Tammet, T. (2012) Information extraction for a tourist recommender system. *Information and Communication Technologies in Tourism 2012:* 343-354. Springer Vienna.

Luberg, A., Tammet, T. & Järv, P. (2011). Smart City: A Rule-based Tourist Recommendation. *Information and Communication Technologies in Tourism 2011:* 51-63. Springer New York.

Popescu, A., Grefenstette, G., Bouamor, H. (2009). Mining a Multilingual Geographical Gazetteer from the Web. *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology* (WI-IAT '09), Vol. 1: 58-65. IEEE Washington.

Popescu, A., Grefenstette, G., Moëllic, P.-A. (2008). Gazetiki: automatic creation of a geographical gazetteer. *Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries* (JCDL '08): 85-93. ACM New York.

Zheng, Y. T., Zhao, M., Song, Y., Adam, H., Buddemeier, U., Bissacco, A., ... & Neven, H. (2009, June). Tour the world: building a web-scale landmark recognition engine. In *Computer Vision and Pattern Recognition*, 2009: 1085-1092.

Acknowledgements

This research has been supported by European Regional Development Fund.

Appendix 3

III

A. Luberg, M. Granitzer, H. Wu, P. Järv, and T. Tammet. Information retrieval and deduplication for tourism recommender sightsplanner. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, WIMS '12, New York, NY, USA, 2012. Association for Computing Machinery

Information retrieval and deduplication for tourism recommender Sightsplanner

Ago Luberg
Eliko Competence Center
Teaduspargi 6/2, 12618
Tallinn, Estonia
Tallinn University of Technology
Ehitajate tee 5, 19086
Tallinn, Estonia
ago.luberg@eliko.ee

Michael Granitzer
Prof. for Media Informatics
University of Passau
Passau, Germany
Michael.Granitzer@uni-passau.de

Honghan Wu
School of Computer and Software, Nanjing University of Information Science & Technology, China
Eliko Competence Centre
Teaduspargi 6/2, 12618
Tallinn, Estonia
honghan.wu@gmail.com

Priit Järv
Eliko Competence Center
Teaduspargi 6/2, 12618
Tallinn, Estonia
Tallinn University of Technology
Ehitajate tee 5, 19086
Tallinn, Estonia
priit@cc.ttu.ee

Tanel Tammet
Eliko Competence Center
Teaduspargi 6/2, 12618
Tallinn, Estonia
Tallinn University of Technology
Ehitajate tee 5, 19086
Tallinn, Estonia
tammet@staff.ttu.ee

ABSTRACT

This paper is about scraping web pages for tourism objects and resolving duplicates for a tourism recommender system Sightsplanner. Gathering information from different web portals, we end up having several versions of the same object in our database. It is very important that we can find out which objects are duplicates and merge those. Only unique objects are presented to the end user. The main focus of this paper is therefore on deduplication problem. We have implemented a duplication detection system and tuned the parameters manually to get up to 85% accuracy. In this paper we present a machine learning setup which we used to improve deduplication accuracy of tourism attractions by 13 percentage points to achieve 98% accuracy. All the steps in the process are presented along with problems we tackled.

Categories and Subject Descriptors

H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Information Search and Retrieval

General Terms

Experimentation, Measurement

WIMS'12, June 13-15, 2012 Craiova, Romania

1. INTRODUCTION

Internet access has made tourism object information available for tourists. A tourist can find several interesting objects she wants to visit during her trip. With all this information, the tourist could have a problem that she cannot find out the best objects for her. We introduce a tourism recommender system named Sightsplanner[1], which helps tourists to find personalized suggestions depending on their interests. In this paper we focus on information retrieval from the Internet. Gathering information from different web portals produces duplicate objects. In this paper we also present how to deal with tourism object deduplication.

The contributions of the paper include:

- Empirical feature study for disambiguating non-frequent geo-objects scraped from different web portals.
- Evaluation of training set selection heuristics in order to tackle highly unbalanced training set in disambiguation problems.
- Optimization of distance functions with machine learning techniques for disambiguation geo-entities.

Our evaluation shows, that we can improve the disambiguation accuracy from a manually tuned similarity measure by 13 percent points to 98% accuracy. Hence, our work shows that machine learning based methods have to be favoured over manually tuning efforts in term of effectiveness and efficiency.

This paper is organised as follows. In section 2 we will give a short overview of the whole recommender system. In section 3 we describe data gathering techniques and give some

[1]Tallinn Sightsplanner, see *http://tallinn.sightsplanner.com/*

Figure 1: GUI first page

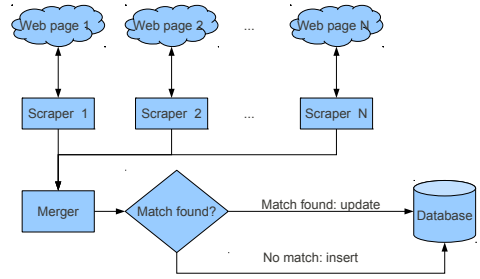

Figure 2: Data scraping

examples about the tourist object data. We will continue with deduplication overview in section 4. In section 5 we present our work on improving deduplication using machine learning. Finally, we give a short overview of related work in section 6, draw a conclusion and discuss future work in section 7.

2. RECOMMENDER SYSTEM

Sightsplanner is a semantic recommender and route composer system for tourists. A tourist can specify her location, time and duration of the visit and her preferences about different types of objects and events. Based on the created tourist profile, the recommender identifies interesting objects for the given user. For each found object a ranking score is found. The objects that the user probably likes have a higher score and vice versa. A planning mechanism organizes the objects and events into a trip timetable. In an interactive feedback cycle, the tourist has the option to modify the suggested trip. To calculate the final list of recommended objects, the following processes are involved: (a) Object verification process, (b) Matching process, (c) Planning process, (d) Result representation process, and (e) Feedback process.

The recommendation process starts when the user opens the web page and defines her interests by using a slider-based approach for adjusting individual preferences. In Figure 1 the first page of the user interface is shown. The tourist can indicate her interest in seven main topics moving the corresponding sliders. Each of them also has subtopics. In the figure the "Events" slider is "opened" and its subsliders are shown. The tourist has stated that she likes "Museums and arts" and "Architecture and city" very much. She also likes some "Eating out", "Shopping" and "Events". Especially she likes "Jazz" and "Rock concerts". Every topic in the user profile matches a type in tourism objects' properties. The user can also specify the start date and time of her visit, visit duration and preferred travelling method.

After the tourist has selected her interests, she starts the overall recommendation process. The user profile is sent to the planner, which is responsible of returning a personalised recommendation for the given profile. The planner uses all the relevant data from the memory database. All the objects in the requested city which also are opened during user's visit will be processed. Based on the object types, location, opening time and some other properties, different trip timetables are created for the user. The planner has to return the best trip it has found after a certain time limit - this is then presented to the user as a timetable and as a map.

The recommended plan can be modified by the user starting an interactive feedback cycle: she can remove some objects or change visit time for objects and re-run the recommendation based on the performed changes: after the modification, the planner takes the changes into account and a new recommendation is created for the user.

3. DATA ACQUISITION

In order to provide accurate recommendations, tourism objects like restaurants and their properties like opening times have to be scraped from web pages and harmonized with already existing data regularly. For Tallinn Sightsplanner, we are scraping six different sites all with different data structure. All those portals are regular web pages. For every data source we have manually described the page structure for our scraper, which then normalizes data into our custom ontology. Gathered information can be divided into more dynamic and more static objects. One-time events (concerts, performances etc.) are more rapidly changing, therefore we are scraping those daily to get changes and new events. Other type of objects are places of interest (POI) with mostly static information - nothing changes in months or even years. POI objects are updated typically once a month.

A brief overview of the architecture of our importer is presented in the Figure 2. The importer can connect to several web pages using manually created scraper algorithms. Each scraper downloads the content of the web pages, finds the necessary information (for example using XPath[2] to extract title, description etc.), normalizes data for our system and sends it to the Merger. We will describe the Merger functionality more in the section 4.

A tourism object, or simply an object, has several facts and zero or more child objects. Every fact is described with following fields: *property*, *value*, *language*, *datatype*, *score*, *source*, *timestamp*. *property* shows what this fact describes (e.g. "title", "address", "phone"). Fact's *value* can be stored in different *languages* and *datatypes* (e.g. number, string, date). A fact also has a confidence or probability score in range $[0; 1]$, where 1 indicates a certain fact and lower value

[2]XML Path Language, see *http://www.w3.org/TR/xpath/*

lowers the certainty of the fact (e.g. 0.7 means that the given fact is true with a probability 0.7). *source* indicates the datasource of the given fact, *timestamp* is date and time when the fact was written.

An object can have child objects. Every child object has the same fields as described previously. In our system we use child objects to describe opening times. One opening time consists of several facts, for example weekday (object is opened on Wednesdays), start time (from 10 AM) and end time (to 7 PM).

Definition 1. The set of all the tourism objects is $\mathbf{O} = \{O_1, O_2, \ldots, O_n\}$.

Possible property URIs are defined by our custom ontology. An object can have several facts with the same property (different titles in different languages etc.). Some of the most important properties for calculating recommendation are "#latitude", "#longitude" (location of the object), "#opening_time" (when is the object opened), "#type" (to match the object against user interests), "#popularity_tourist" (to favour more popular objects), "#visit_time" (the time suggested to spend on sight). Some other properties are important for presenting the object to the user. The most important ones are "#title" and "#description", which will be stored in different languages. An object can have a different title in English and in Estonian. Different values are presented to the user depending on their user interface language. Often web pages do not provide data in separate languages. We try to use automatic translation via Google Translate API [3] to fill in the missing values. We make use of the *score* field and have lower value for automatically translated texts. Usually scraped information from the source web page gets a score 1.0. For automatically translated title and description we will use lower score value (e.g. 0.5). Later, when we find the same object from another website, where missing information is available, we can prefer a title fact with higher score over the automatically translated one.

To recommend an object to the user, the object needs to have the same category that the user has marked as her interest. Often data sources do not define the type or the category of the object. Or categories they are using do not match with our set of categories. To overcome this problem, we are using keyword extraction from the description field in different languages to find keywords which could be mapped into a category known for us. For every language we have a list of keywords along with score values. The score value will be applied to the created category fact. For example, if the system finds a keyword "painting" in the text and in our keyword list the "painting" keyword is mapped to "#art" category with a score 0.6, then the object will get a fact, which says that the object has a category "#art" with a confidence score 0.6. The score for category facts can be also seen as a strength or a weight. More details about keyword extraction and data in general is presented in [4].

4. DEDUPLICATION

Scarping information from different sources yields in duplicate objects as most of the web pages do not provide global URI, which could be used to match the same resources from different sources. From the recommendation point of view, having duplicate objects in the proposed schedule reduces the quality of our system. Often sources do not have full information about an object. Different source may have some information about the object which was missing in the first source etc. So, merging duplicate objects may improve the quality of the merged object. Taken into account that deduplication will improve our recommendations and object information quality, we were motivated to work on the solution.

First, we will define some notations which we will be using later.

Definition 2. Object A_i is a duplicate of object A_j if they represent the same pyhsical object. Let d be a symmetric function which returns 1 if all its arguments are duplicates, 0 otherwise:

$$d(A_1, \ldots, A_n) = \begin{cases} 1 & \text{if all } A_i \text{ are duplicates} \\ 0 & \text{otherwise} \end{cases}$$

In addition to comparing different objects, in our notation object is a duplicate of itself: $d(A, A) = 1$ or $d(A) = 1$.

Definition 3. A group of duplicates, called a *cluster*, is a set C which consists of at least one object from all the objects $\mathbf{O}$ so that all the included objects are duplicates:

$$C = \{A_1, \ldots, A_n | d(A_1, \ldots, A_n) = 1\}$$

A cluster could also consist of only one object $C = \{A\}$, as $D(A, A) = 1$.

Definition 4. A maximal group of duplicates, called a *maximal cluster*, is a cluster which cannot accept any new objects so that all the objects would be duplicates of each other (there is no additional object which is a duplicate of the objects in the cluster).

In this paper, we are interested in *maximal clusters*. Therefore we use term *cluster* to denote *maximal clusters* if not noted differently.

Definition 5. A similarity between two objects A and B is defined by function S which is weighted average over similarity values:

$$S(A, B) = \frac{\sum_i w_i * sim_i(A, B)}{\sum_i w_i}$$

sim_i is a function which compares certain property or properties of two objects and returns a similarity score in range [0; 1]
w_i is a weight value for the similarity function sim_i.
The value of S will be in range [0; 1].

Definition 6. Two objects are duplicates by similarity function if the similarity function S between the objects A and B exceeds a threshold T:

$$S(A, B) \geq T \Rightarrow d(A, B) = 1$$

In the data acquisition section we gave an overview of the data importing process. We also mentioned the Merger component, which deals with deduplication. Every scraped object is sent to the Merger which compares the new object with the existing ones by computing the similarity value between the objects (using title similarity, location similarity

[3]Google Language API Family, see *http://code.google.com/apis/language/*

etc.). From certain similarity value threshold, the new object is considered to be a duplicate of the found existing object(s). In the case of match, the existing object in the database is updated with new scraped data. If there is no matching object in the database, the scraped object is added as a new object. More details about the merging and Merger implementation are presented in [5].

For our current system, we have manually tuned weights w_i for similarity functions sim_i and the threshold T value. Our best setup had equal weights for every similarity function except for the distance similarity, which had double importance. We used total of 6 different similarity functions. The threshold value which indicated the separation of duplicate and non-duplicate object pair was 0.9. Using this setup, we were able to get F-score 0.85. Our goal was to improve this metrics at least above 0.9. That is where we started using machine learning.

5. EXPERIMENTS

Instead of manually trying to adjust the weight parameters of similarity function in definition 5, we have used machine learning to find the best settings. In this section, we will present the setup of the whole process along with the results. The section is divided into subsections about learning problem, data used for experiments, feature selection, sample selection, learning setup and results.

5.1 Learning problem definition

Instead of trying to group objects directly, we have a different approach. We try to learn whether two objects are duplicates or not based on the property similarity functions. Every sample in our learning set is a set of similarity values calculated by comparing two objects. That is the reason, why we presented object pair counts in Table 1. We want to learn how to separate positive and negative pairs (e.g. whether a pair of objects represent the same physical object or not). If we have 4 objects A, B, C, D, we can have 6 unique samples: comparisons of pairs AB, AC, AD, BC, BD and CD.

Definition 7. Given the similarity function $S(A, B)$ (definition 5) we define $\mathbf{f_{A,B}}$ as a feature vector with f_i being the evaluation of sim_i on objects A, B.

$$\mathbf{f_{A,B}} = \langle f_1, f_2, \ldots, f_n \rangle$$

where

$$f_i = sim_i(A, B)$$

sim_i may be chosen from a set of similarity functions Sim.

A sample is given by $\mathbf{x_{A,B}} = \mathbf{f_{A,B}}$
A sample can be also called $\mathbf{x_i}$ if the compared objects are not known or not important.

A sample set
$X = \{\mathbf{x_{A_i,A_j}} | A_i, A_j \in \mathbf{O}\}$ or
$X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$, where m is the number of samples.

A classification label is given by

$$y_{A,B} = \begin{cases} 1 & \text{if } d(A, B) = 1 \\ 0 & \text{if } d(A, B) = 0 \end{cases}$$

A label set
$Y = \{y_{A_i,A_j} | (A_i, A_j \in \mathbf{O}), (\mathbf{x_{A_i,A_j}} \in X)\}$ or
$Y = \{y_1, y_2, \ldots, y_m\}$, where m is the number of samples. Label set Y has labels for the same object pairs which are present in sample set X in the same order.

A sample $\mathbf{x_{A,B}}$ is called a positive sample if $y_{A,B} = 1$, e.g. object A and B are duplicates. If $y_{A,B} = 0$, then the sample $\mathbf{x_{A,B}}$ is called negative sample.

A learning function is given by $\lambda = \langle L, X, Y \rangle$, where L defines a learning algorithm along with parameter values, X is a set of samples, Y is a set of labels for the samples X.
Learning function λ returns a function p which predicts with some accuracy whether given two objects A and B are duplicates or not.
The goal of λ is to find a function p which would yield in highest prediction accuracy.

Our learning problem is defined to learn, whether given two objects are duplicates or not. One sample (which can used for training and also for testing) is a set of similarity values between two objects. Every similarity value indicates a similarity of a certain property or properties between two objects. For example, let us use two features: title similarity and description similarity. Every sample in our training/testing dataset would have 2 values. If we have objects A and B, which are very similar, then the sample can have values $\mathbf{x_{A,B}} = 1.0, 0.9$. Another pair of objects A and C might not be similar and have values $\mathbf{x_{A,C}} = 0.3, 0.1$. Both those samples have also a label value: $y_{A,B} = 1$, $y_{A,C} = 0$. If the system would have only those two samples for training, it might learn that high feature values will yield in label 1 and vice versa. This is actually correct, because our features all have built this way that similar objects (near-duplicates or duplicates) have high values.

We use different sample and label sets for training and testing. After we have used some samples and labels for training, we want to test how well we have trained our model. For example, we have samples $\mathbf{x_{B,C}} = 0.5, 0.9$ and a label $y_{B,C} = 0$. After we have applied learning algorithm to our training set, we have got our deduplication function p. As we saw before, high feature values seem to indicate that two objects are duplicates. It may happen, that our function predicts $d(B, C) = 1$. The real label is $y(B, C) = 0$. We can see, that we made an error and may-be we did not learn the best possible deduplication function. Depending on the setup, we can just accept that we have made an error or we can try to train a better model. More about the learning setup will be presented in one of the next subsections.

5.2 Data for learning

Our initial Tallinn dataset had objects from different categories. Some data sources only provide eating places, some only events and so on. The dataset only has a small number of duplicate objects and maximum number of different data sources for one objects is three (there are three sources which provide eating places, so the same restaurant can be scraped from three sites). Therefore we have scraped a separate dataset for our learning setup. The dataset consists of Tallinn eating places from five different web portals. Some eating places are present in every data source, so after merging, the objects will be merged from five different sources. In addition to Tallinn dataset, we are using Riga tourism objects to test our learning model. The Riga dataset is not limited to eating places, there are also museums, galleries etc. Testing our trained model on Riga dataset gives us in-

formation whether our solution of duplicate detection can be applied for cross-city and cross-category datasets. Riga dataset is scraped from two different sources.

To be able to train and evaluate learning algorithms, we have manually annotated all the duplicate pairs. We have created a simple web interface which allows us to manually compare every objects with another object. Based on the information available, a user was able to state whether two objects are physically the same or not. Tallinn data consists of about 1800 scraped object, comparing every object with every other object would yield in 1.6M comparisons. To limit this number, we have added a duplicate candidate selection, where only a certain number of best matching objects are presented to the user. Objects, which are too far away from each other most likely are not duplicates. We also filter objects by type and data source. Data source filter means that we compare an object only with objects from different data sources. Our initial presumption was that a data source does not have duplicates within their data. It comes out, that actually there are duplicates within certain sources, but this is not a problem. After we have found that two compared objects are duplicates, then the connection is bidirectional (if object A is a duplicate of object B, then object B is a duplicate of object A). Every group of duplicate objects forms a complete graph (if object A is a duplicate of object C and object B is a duplicate of object C, then object A is also a duplicate of object B). Because of the completeness, it is usually enough to compare an object only to objects which do not have the same source. With filtering, every object had average about 20 possible candidates, which narrowed down the comparison space about 100 times.

Manual annotation still raised many questions amongst users who had to find duplicate pairs. Even if you are local and know most of the tourism objects, there are still cases, which cannot be solved with 100% confidence. Also, as we mentioned, we used candidate selection, which might have left some duplicates out. Objects, which were not in the candidate list, were not checked by the annotator. Altogether we believe that the error of manual deduplication can be about 3-5%. We will take this into account when we later evaluate our results.

Table 1: Statistics about the dataset for Tallinn and Riga

Property	Tallinn	Riga
Object count	1808	3839
Different sources	5	2
Non duplicates	478	3762
2-object groups (object count)	203 (406)	75 (150)
3-object groups (object count)	133 (399)	1 (3)
4-object groups (object count)	68 (272)	-
5-object groups (object count)	43 (215)	-
6-object groups (object count)	5 (30)	-
8-object groups (object count)	1 (8)	-
Positive (duplicate) pairs	1543	78
Negative (non-duplicate) pairs	1.6M	7M
Positive pair %	0.1 %	0.001 %

We have presented an overview of the dataset in Table 1. For Tallinn dataset, we have total of 1808 scraped objects from 5 data sources. 478 objects did not have any duplicate objects (or we could say they form up 478 clusters each consisting of only one object), 406 objects formed 203 groups with 2 duplicates in each cluster, etc. As can be seen, some objects have duplicate entries in the same data source. For example, in Tallinn dataset there is one cluster which is merged from 8 initial objects (duplicate object was present once in 2 data sources and twice in 3 data sources). Total number of unique duplicate pairs (graph undirected edges) is 1543. If object A is a duplicate of object B, then it is counted only once - pair object B is a duplicate of object A is not counted. All other possible unique pairs between the objects are non-duplicate pairs. For Tallinn data, there are about 1.6M non-duplicate pairs. We have also presented a percentage of duplicate pairs to non-duplicate pairs to indicate the balance of our dataset. As can be seen, for Riga the percentage is even worse.

5.3 Feature selection

We have implemented about 20 different functions for features. The most interesting for this paper are:

- Title comparison using Levenshtein distance[4];
- Custom title comparison with weighted words (common words weigh less and therefore do not change the outcome too much);
- Custom title comparison, which we will describe below;
- Custom title comparison with weighted words (common words weigh less and therefore do not change the outcome too much);
- Euclidean distance using originally scraped coordinates;
- Address string comparison;
- Euclidean distance using coordinates which were calculated from address strings.

Custom title comparison works as follows. The title is split into words. For every word a match from other title's words are found. Every match gives a positive score. All the words, which do not have a match in other title, will give negative score. Both scores are added together and normalized (to get a result within the range [0, 1]) and the result is the similarity between object titles. To find a match for a word, the words do not have to match exactly. A certain number of symbols can be different depending on the length of the word. If all the words from one title are found in the second title, then similarity is near to 1 (for example "papa pizza" vs "papa pizza with some extra words" will yield in almost total similarity, as additional words do not make the match worse). As we mentioned earlier, an object can have different titles in different languages. All the languages are compared and the best match is returned. The same logic is applied with weighted words. More common words have less impact on both positive (in case of match) and negative (in case of no match) score. If we compare titles like "Papa pizza" and "Mama pizza", then "pizza" is common word and having a common word in both titles does not make objects similar (positive match will be low). In the previous example, words "mama" and "papa" are probably

[4]Levenshtein distance, edit distance, see *http://en.wikipedia.org/wiki/Levenshtein_distance*

not so popular, which makes the negative match high. In this example, objects are not similar. But if we consider titles like "Papa pizza" and "Papa restaurant". "Papa" is not very popular name, so the positive match is high. "Pizza" and "restaurant" are usually very popular, which makes the negative score low. Based on the weighted title comparison, those two objects could represent the same physical object.

To give a short example of custom title comparison, let us consider two titles "Tallinn city hall" and "Tallinn city pharmacy". There are two words, which are present in both titles: "Tallinn" and "city". Both titles have one additional different word ("hall" and "pharmacy"). If we do not use weights, we find an average non-matching word count for both titles. In this example the average is 1. The similarity between two titles is calculated as follows: *matching_count* / (*matching_count* + *avg_non_matching_count*). In our example, it would be 2 / 3 = 0.66. If we consider titles like "papa pizza" and "papa pizza in shopping center", then matching word count is 2. We also have 3 non-matching words. All the non-matching words which are present due to title length difference (if one title has 2 words less than the other, then those 2 words usually are also non-matching) have lower penalty. In the current example, the calculation can be for example: 2 / 2.6 = 0.77. The lower penalty depends on the count of non-matching words and on the length of the titles. If the titles are long, then the penalty will get lower (if you have one non-matching word for 10 words title, then penalty will be near to 0).

In addition to mentioned custom title comparison, we also have so called custom title comparison with join. Regular custom title comparison splits the title into words and starts comparing. The comparison with join tries to combine different words into one and run the custom title comparison then. For example, if we have titles "McDonalds" and "Mc Donalds", then the regular custom title comparison would return 0 as there is no matching words, whereas after joining two words in the second name we would compare the same titles and the result will be high score. The join version of comparison basically finds all the combination of joining consecutive words on both titles and for every combination, the regular custom title comparison is run. This makes the join version several times slower.

To find out, which combination of features is the best, we have done training and testing with all the possible combinations up to 5 features. We will present results soon. First we have to discuss the sample selection.

5.4 Sample selection

As we presented in Table 1, the possible number of object pairs is for Tallinn data about 1.6M. The table also shows that only 0.1% of those positive samples (duplicate pairs). If we would take all the samples, then we would have several problems:

- Generating a feature value set for 1.6M pairs takes time;
- Learning with large number of data takes a lot of time;
- The balance between positive and negative samples is heavily skewed.

One of our goal is to use as few samples as possible. Therefore we did not try to use all the samples. Instead, we aimed for 10 000 samples for Tallinn data. If we used random sample selection, we would end up only about 15 positive training samples, which is obviously too few.

If we think about the possible samples in our dataset, then for one object, there are about 1800 samples available (comparison with every other object). Feature like objects distance will have similarity value 0 or very low when the distance is more than 500 meters for example. Most of the objects are farther than this. For distance features, maximum 100 objects would give similarity score about 0. The same is usually the problem with titles. For example, an object with title "McDonalds" do not have many matches amongst the whole dataset. If we would take all 1.6M negative samples, then many will end up having all the feature values near to zero. We do not need to include all those for our learning dataset. Instead, we are more interested on negative pairs, which are closer to duplicates. With positive samples, we do not filter anything out - we will use all 1543 positive samples for learning.

Our sample selection for Tallinn data currently has 10 000 samples, 1543 of those are positives, and we try to take mostly negative pairs which have high feature values (near-duplicates). Of course, we cannot leave out the low negative samples (all feature values near to zero), otherwise we might end up with classification which only recognizes mid-values as non-duplicates. About 1000 samples have close to zero feature values. For Riga dataset, we have taken more samples. We have just limited the number of samples with feature values close to zero to 10 000. Riga data is used only for testing and it has about 100 000 samples: 78 positive ones, about 10 000 low negative ones, the rest is mid-valued or near-duplicate negative samples.

5.5 Learning setup

We have used Python software scikit-learn[5] to assist our learning process. The software supports various number of different learning algorithms. For our problem, we have used SVM (Support Vectore Machine) classification and decision trees. For SVM, we used grid search, which tries several different parameters and returns the one with the best results. The grid search tries both linear and radial basis function (RBF) as a kernel. For decision trees, we are using extra trees which train several (30 in our case) independent models randomly and uses average over the models to predict. B learning algorithms are run with all the combination of all the features. This way we can find out the most important features. We also would like to minimize the calculation costs for prediction, therefore we try to minimize the number of features necessary for a model.

As mentioned earlier, we use Tallinn data for training and testing, Riga data is used only for evaluation. Tallinn dataset is divided into two equal sized parts where the ratio of positive and negative samples also remains the same. One part is for development and the other is for evaluation. On development part, we do training with 10-fold cross-validation. The model which yields in best results on cross-validation, will be used for evaluation both with remaining Tallinn data and with Riga data.

5.6 Learning results

We have constructed several different datasets (different number of samples and different selection of samples) which

[5]scikit-learn: machine learning in Python, see *http://scikit-learn.sourceforge.net/stable/*

Table 2: Learning results using one feature

Feature (code)	Algorithm	Tallinn			Riga		
		precision	recall	f-score	precision	recall	f-score
Address string comparison (ADD)	rbf	0.92	0.94	0.93	0.52	0.91	0.66
	extree	0.96	0.92	0.94	0.59	0.88	0.71
Title comparison without joining words (T1)	rbf	0.99	0.85	0.91	0.76	0.91	0.83
	extree	0.99	0.85	0.91	0.65	0.91	0.76
Title comparison with joining words (T2)	rbf	0.98	0.89	0.94	0.69	0.96	0.80
	extree	0.98	0.92	0.95	0.52	0.97	0.68
Title comparison with edit distance (ED)	rbf	0.96	0.85	0.90	0.09	0.94	0.16
	extree	0.97	0.86	0.91	0.14	0.91	0.24
Distance with original source coordinates (OD)	rbf	0.76	0.67	0.71	0.65	0.67	0.66
	extree	0.84	0.82	0.83	0.36	0.90	0.51
Distance with re-calculated coordinates (RD)	linear	0.93	0.95	0.94	0.55	0.96	0.70
	extree	0.96	0.95	0.95	0.71	0.88	0.79
Title comparison without joining words, with word weights (TW1)	linear	0.97	0.90	0.94	0.36	0.99	0.53
	extree	0.97	0.96	0.96	0.27	0.99	0.43
Title comparison with joining words, with word weights (TW2)	linear	0.97	0.94	0.95	0.28	1.00	0.43
	extree	0.98	0.97	0.98	0.28	0.99	0.44

Table 3: Learning results using two features

Feature codes	Algorithm	Tallinn			Riga		
		precision	recall	f-score	precision	recall	f-score
ADD + T1	rbf	0.99	0.95	0.97	0.93	0.99	0.96
	extree	0.99	0.97	0.98	0.73	0.99	0.84
T1 + RD	linear	0.98	0.97	0.98	0.93	1.00	0.96
	extree	1.00	0.97	0.98	0.93	0.99	0.96
RD + TW1	linear	0.97	0.99	0.98	0.71	1.00	0.83
	extree	1.00	0.98	0.99	0.88	1.00	0.93
RD + TW2	linear	0.99	0.97	0.98	0.80	1.00	0.89
	extree	1.00	0.99	0.99	0.89	1.00	0.94
T2 + RD	rbf	0.98	0.99	0.98	0.86	1.00	0.92
	extree	1.00	0.98	0.99	0.94	0.99	0.96

Table 4: Learning results using two features

Feature codes	Algorithm	Tallinn			Riga		
		precision	recall	f-score	precision	recall	f-score
ADD + T1 + RD	rbf	0.99	0.97	0.98	0.97	0.99	0.98
	extree	1.00	0.97	0.99	0.93	0.99	0.96
ADD + T2 + RD	linear	0.99	0.98	0.99	0.94	1.00	0.97
	extree	1.00	0.99	0.99	0.93	1.00	0.96
T1 + RD + TW1	linear	0.99	0.97	0.98	0.91	1.00	0.95
	extree	1.00	0.98	0.99	0.91	1.00	0.95
T1 + RD + TW2	linear	0.99	0.99	0.99	0.89	1.00	0.94
	extree	1.00	0.99	0.99	0.94	1.00	0.97
T1 + RD + T2	rbf	0.98	0.99	0.98	0.87	1.00	0.93
	extree	1.00	0.98	0.99	0.96	0.99	0.97
ED + RD + T2	linear	0.98	0.99	0.98	0.91	1.00	0.95
	extree	1.00	0.99	0.99	0.92	0.99	0.95

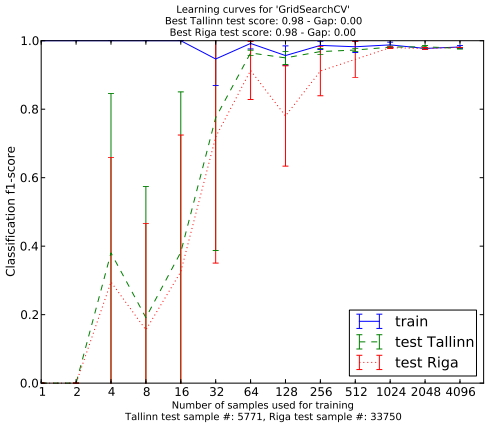

Figure 3: Learning curves for training, Tallinn test and Riga test data using grid search (SVM parameter optimization) and features ADD + T1 + RD

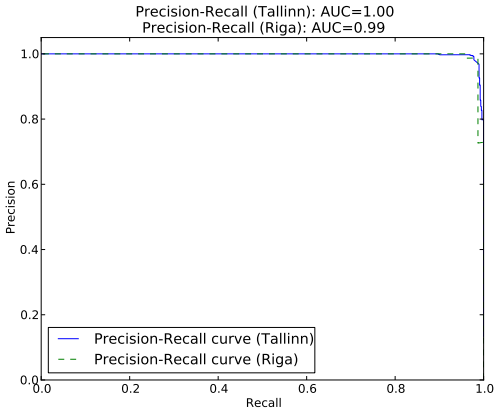

Figure 4: Precision-recall curve for features ADD + T1 + RD trained with SVM

were trained with different learning algorithms. The total number of test run is over 400. Here we present results for some of those tests. The tables described in this section all have the same structure. Each table has different number of features used for the learning problem. The first column indicates, which features are used for training and testing. Every feature set is evaluated with 2 different learning algorithms. The first one is Support Vector Machine (SVM) and the other is extra trees (extended version of decision trees). For SVM, we have shown the used kernel (linear or rbf - radial basis function). All this data is trained with the same amount of randomly chosen samples from Tallinn data. The ratio of positive and negative samples remains the same for training and testing data. All the tables present results which are trained with 50% of Tallinn data (5771 samples, 771 of those are positive). The other part of Tallinn data will be used for evaluation. In addition, we have evaluated every trained model with Riga data (33750 samples, 78 of those are positives).

Metrics used for evaluation are:

$$precision = \frac{tp}{tp + fp} \tag{1}$$

$$recall = \frac{tp}{tp + fn} \tag{2}$$

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3}$$

,

where tp is the number of *true positive* predictions (how many predicted positive samples are actually positives), fn is the number of *false negative* predictions (predicted negative, but actually are positive), fp is the number of *false positive* predictions (predicted positive, actually are negative). In our problem, we concider both precision and recall equally important, therefore we use F_1 score[6] as the main evaluation of our model.

The Table 2 shows the results for 8 best features used alone. For training and testing, only values of one feature (for example title similarity) was used. Note that every feature has a short code after the name. Those codes are used later in other tables. From the results we can see that using only one feature, we can have F-score near 0.8. We can also see that edit distance or Levenshtein distance (feature ED) alone does not separate duplicates from non-duplicates very well.

The Table 3 present results for some combination of two features. The F-score here is already very promising. We can see here that title similarity combined with location/address similarity gives good results. For example ADD + T1 (address string similarity and custom title similarity) and T1 + RD (custom title similarity and re-calculated distance similarity) both give F-scores about 0.95. We did test also with the combinations of three features which are presented in Table 4. All presented results have F-score near or over 0.95. The first feature set ADD + T1 + RD (address string similarity, title custom similarity and recalculated distance similarity) has the highest Riga test result. Also, as we stated earlier, we believe that the manual annotation of duplicate objects has also an error of 3-5%, then 0.98 F-score

[6] F_1 score treats both precision and recall equally important

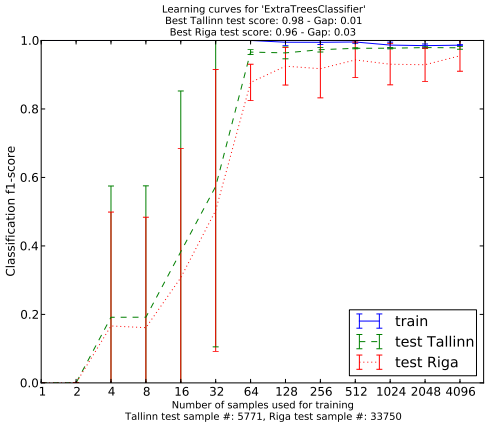

Figure 5: Learning curves for training, Tallinn test and Riga test data using extree (decision trees) and features ADD + T1 + RD

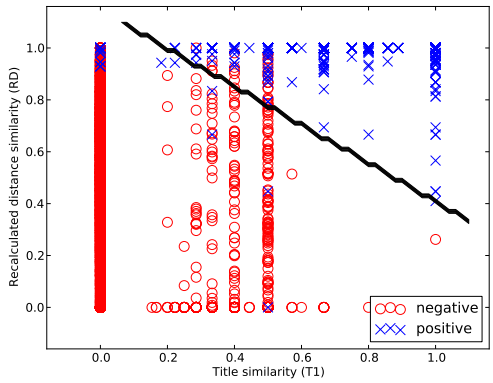

Figure 6: Tallinn dataset with features T1 + RD

is very good result. We have completed tests also for 4 and 5 feature combinations, none of those performed better than 3 feature combination ADD + T1 + RD.

In addition, we evaluated the learning capability of our best model. In the Figure 3 we have drawn learning curves (which indicate F-scores) for training and testing datasets for different training dataset size using grid search (SVM). To give a better overview of the changes to the curves, we have used logarithmic scale. X-axis shows the training dataset size. Our results in the tables were trained with about 5000 samples. As can be seen from the learning curve, we get good predictions already starting from 64 training samples. The learning curve also shows that our learning problem does not suffer from overfitting (our training accuracy is not too high compared to testing accuracy, the gap is close to 0 starting from 1000 training samples).

If we compare grid search learning curve with decision trees' curve in Figure 5, we can see, that decision tree takes more training samples to get better result on Riga (additional test) data. In addition to learning curve, we have plotted precision and recall curve in Figure 4 for SVM model with features ADD + T1 + RD.

In the end, we will give an example plot of our samples with 2 features in Figure 6. We have used SVM to plot features T1 + RD (custom title similarity and recalculated distance similarity) which gives an idea, how the features are located. The black line is class trained separator: right upper corner is for positive (duplicates) predictions, lower left corner is for negative (non-duplicates) predictions.

6. RELATED WORK

Deduplication is very popular topic, especially when we have access to more information. Various data sources across the Internet are presenting information about the same objects. Semantic web tries to ease the problem to offer global URIs or references to widely known information sites (for example DBpedia[7]). Here we will look into some papers, which have been inspired our work and will continue to do that.

One similar system to ours is presented in [7]. They also have geographical objects and they need to detect duplicates. They have introduced 4 similarity functions or features: two title comparisons, address similarity and category similarity. Taking category into account is something we have to look into. The main problem is that every data source has its own categories. We do keyword extraction to find some types for every object, but often we will get misplaced objects in some category. Another problem is that currently we have build category hierarchy for user interface. Often this does not work well in terms of similarity. We should introduce a separate category model, which could be used for finding duplicate objects.

Aforementioned article has surprisingly low accuracy after the training. May-be the data we are using has more quality and therefore learning and predicting work even with small dataset sizes. The authors do not go into details about the learning part. Also, it is somewhat unclear, why they did not use more samples for their training and testing set.

Bilenko and Mooney have written about deduplication in databases [2] and evaluating duplicate detection [3]. The approach they are using is similar to what we have described in this paper. One interesting work from them is learning string distance metrics. While in our case, edit distance did not give good enough results, we manually created a custom title comparison function. Instead, we could also learn, which metrics could be used and combine different standard methodologies with weights in order to find the best one for tourism object titles. As most of our objects do not have descriptions, we cannot make use of using larger text to compare objects. This would be interesting and hopefully we will start getting more descriptions and reviews about the objects, which gives us opportunity to do more text similarity calculations. One very active field of study is research paper deduplication. The text of the paper is very important to detect whether two articles are the same or not.

In [3], they have mentioned precision and recall curve as one evaluation metrics for deduplication problem. We have also drawn those curves to study features and dataset. In

[7]DBpedia, see *http://dbpedia.org/*

the result section we presented the curve for our best model. With current data and feature setup, we were not able to use this curve much, because our problem was too easy to solve. We will be applying our deduplication methodology on another domain, where we certainly will have lower accuracy. That is where the the precision and recall curves can help to solve problems.

In [6] the author represents learning approach for duplicate detection for geospatial objects. The approach is again very similar to what we have presented here. They argue in their paper that traditional textual similarity functions do not work well with place names, because their stylistic variability is too great. Our custom title similarity function is implemented exactly because of this. They also mention that common names (like *street*) could be ignored when comparing address strings. We had one feature, which tried to do that (more specifically, words that occur more often, have less impact). But non weighted word similarity worked better. We are in the middle of analysing the results, where we can see, in which cases weighted words would outperform non-weighted words, if at all. The results the author presents in the paper are very promising. The paper has a good overview of text comparison metrics, which we could also try on our system to see, whether we can get better results for title comparison.

Martins describes the problem of different geocoding, which we also tackled in our research. We try to get geo coordinates for every object in our database to ease to route planning. Also, the same coordinates can be used to find duplicates. If we scrape information from different web sites, we end up with noisy data. Sometimes the same object was located 500-1000 meters away. We used normalization of geo coordinates using object addresses, which improved the quality. In the result table with one feature, you can compare features OD (originally scraped coordinates) and RD (recalculated based on address). After recalculation, if the address was the same, we actually got the same coordinates for the objects. Before, there was always some difference. For some objects, it was 5-10 meters, for other 100 meters etc.

An article about Swoosh [1] presents good results in entity resolution problem in terms of accuracy and speed. They introduce an algorithm which can reduce the number of comparisons between the objects. While our system basically compares every object to every other object, their algorithm can perform magnitude of less comparisons. To determine the similarity between two objects, they use features and coefficients/thresholds. Manual tuning of those coefficients can give good results, but as we learned in our system, we were able to improve the accuracy when we learned from the data itself. Mentioned algorithm Swoosh is still very promising and we are planning to do some more experiments where we integrate Swoosh with machine learning. The idea would be to learn which features and which coefficients to use, while Swoosh can help with finding candidate objects and merging.

7. CONLUSIONS

In our paper we have given an overview of a tourism recommender system Sightsplanner. We described the information gathering process and presented our manually tuned deduplication algorithm. From there we were motivated to improve our accuracy of duplicate detection. We have used machine learning techniques to find ways to increase our accuracy. In this paper we describe, how we have set up the learning system. We have manually annotated 1500 duplicate objects to evaluate our learning. We introduced the technique, how we compare objects, where every sample in our dataset is a comparison of two objects. Using pairs like that, we are able to find groups of three or more duplicate objects.

We present some of our results in the previous section. Along with prediction accuracy, we also analysed learning curves. As can be seen from one of our learning curves, we could use several times less training data to achieve approximately the same f-score. Depending on the needs of the system. To guarantee f-score 0.9 or higher, we could use may-be 10 times less training data. This gives an opportunity to start with training your model with rather small dataset. The learning curve also showed that with adding more data, the f-score may even improve (if we start for example 256 training samples).

Another finding of our work is that our approach learning by comparing tourism object pairs can be applied cross-category, cross-language and cross-city datasets. For training, we used Tallinn eating places. For additional evaluation we tested trained models on Riga data, where in addition to eating places there are museums, galleries etc. The results show that using two or more features, we are able to get very high results with Riga data.

The main goal of our learning setup was to improve the manual deduplication accuracy. Our manually tuned algorithm was able to detect duplicates with F-score 0.85. With machine learning, we have increased the f-score to **0.98**.

In the future, we will try to apply the same methodology used in tourism domain to some other domain, probably for research paper deduplication. We have to work on our textual similarity functions. We also plan to run our recommender system for larger cities, where we can gather more data to further evaluate our deduplication system. Along with more data, we will investigate new features (we already mentioned category similarity).

8. ACKNOWLEDGMENTS

This research has been supported by European Regional Development Fund. This work has been funded by the European Commission as part of the FP7 Marie Curie IAPP project TEAM (grant no. 251514).

9. REFERENCES

[1] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal*, March 2008. VLDB Journal (Online First) link: http://dx.doi.org/10.1007/s00778-008-0098-x.

[2] M. Bilenko. Learning to combine trained distance metrics for duplicate detection in databases. *Submitted to CIKM-2002*, (February):1–19, 2002.

[3] M. Bilenko. On evaluation and training-set construction for duplicate detection. *of the KDD-2003 Workshop on*, pages 7–12, 2003.

[4] A. Luberg, P. Järv, K. Schoefegger, and T. Tammet. Context-aware and multilingual information extraction for a tourist recommender system. In *Proceedings of the 11th International Conference on Knowledge*

Management and Knowledge Technologies - i-KNOW '11, number c, pages 1–8, New York, New York, USA, 2011. ACM Press.

[5] A. Luberg, P. Järv, and T. Tammet. Information Extraction for a Tourist Recommender System. In *Information and Communication Technologies in Tourism 2012*. 2012.

[6] B. Martins. A Supervised Machine Learning Approach for Duplicate Detection over Gazetteer Records. In C. Claramunt, S. Levashkin, and M. Bertolotto, editors, *GeoSpatial Semantics*, volume 6631 of *Lecture Notes in Computer Science*, pages 34–51. Springer Berlin / Heidelberg, 2011.

[7] Y. Zheng, X. Fen, X. Xie, S. Peng, and J. Fu. Detecting nearly duplicated records in location datasets. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 137–143, New York, NY, USA, 2010. ACM.

Appendix 4

IV

A. Luberg, P. Järv, and T. Tammet. Information extraction for a tourist recommender system. In M. Fuchs, F. Ricci, and L. Cantoni, editors, *Information and Communication Technologies in Tourism 2012*, pages 332–343, Vienna, 2012. Springer Vienna

Information Extraction for a Tourist Recommender System

Ago Luberg[a,b], Priit Järv[a,b] and Tanel Tammet[a,b]

[a] Eliko Competence Centre
Tallinn, Estonia
ago.luberg@eliko.ee

[b] Tallinn University of Technology
Tallinn, Estonia
tammet@staff.ttu.ee
priit@cc.ttu.ee

Abstract

We will present the information extraction algorithms for a semantic personalised tourist recommender system Sightsplanner. The main challenges: information is spread across various information sources, it is usually stored in proprietary formats and is available in different languages in varying degrees of accuracy. We will address the mentioned challenges and describe our realization and ideas how to deal with each of them: scraping and extracting keywords from different web portals with different languages, dealing with missing multi-lingual data and identifying the same objects from different sources.

Keywords: recommender system; information retrieval; entity disambiguation

1 Introduction

We will describe the information retrieval techniques used in the tourist recommender system Sightsplanner for Tallinn (http://tallinn.sightsplanner.com, [2011]). The first iteration of the system was called Smart City and is presented in (Luberg et al., 2011). We use semi-automatic web scraping of different web portals to gather information about tourism objects. The data is transformed into our own custom ontology and cleaned up for later use by the recommender.

Information about the objects on the internet is spread across various sources and is usually stored in different languages. The outcome shown to a tourist should be complete and precise facts about the sights. The paper focuses mainly on how the data about the tourism objects is gathered, how it is stored and how it is cleaned up. We will present both the problems we have overcome and the problems we still have to tackle to improve the results of our data gathering module.

In the following two sections we will provide a brief overview of the system. In the section 4 we will describe data retrieval process in more detail. We will end our paper with related work and conclusion.

2 Tourism Recommender System

Sightsplanner is a semantic recommender and route composer system for tourists. A tourist can specify her interests on different categories, her location and time of the visit. Provided characteristics create a user profile, which is sent to the recommender. Based on the profile, the system calculates a ranking score to every object in the database. The objects that the user likes more will get a higher score and vice versa. A schedule is created out of the objects which will give the best overall score. Several processes are involved when calculating the final list of objects: object verification process, matching process, planning process, result representation process, and feedback process.

The recommendation process starts when the user opens the web page and provides her preferences. A slider can be used to define the strength of interest on every object category. In our system we are using two levels of categories. One general category (like "*Music*") has subcategories (like "*Rock*", "*Jazz*"). Every object in the database has also defined types from the same set of categories. Besides interests the user can define her start time, duration of the visit and preferred travelling method.

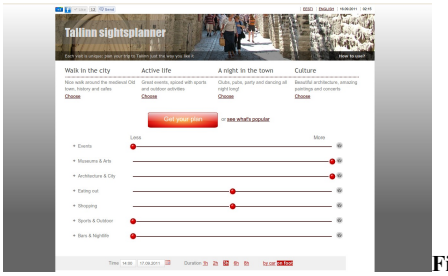

Fig. 1. A screenshot of the Sightsplanner.com frontpage.

After the user has submitted the form, the created profile is sent to the recommender. The planner filters out objects by location, opening times and categories. Found objects are combined into a timetable iteratively. After a certain time limit the system returns the result with the highest total score which is presented to the user as a timetable and a map.

The user can modify the result by giving feedback for objects: objects can be removed and the visiting duration of objects can be changed. A new planning process is started after the submission of the modifications.

3 Technical Overview

The main components of the system are presented in the Figure 2. In this section we will describe data discovery and recommendation process with details about each of the components.

Information about the tourism objects stored in our database is gathered from various web sources in proprietary formats and is available in different languages in varying degrees of accuracy. The more detailed and coherent we can gather this data, the better recommendations we can provide.

In the current implementation we scrape information about objects in a semi-automatic fashion. First we crawl different web portals and normalise the gathered data into our custom ontology. In some cases the information available is not complete, therefore we have added an option to modify or add data manually. All the gathered information is stored in PostgreSQL (www.postgresql.org, [2011]) database. In section 4 we will describe the data discovery process in more details.

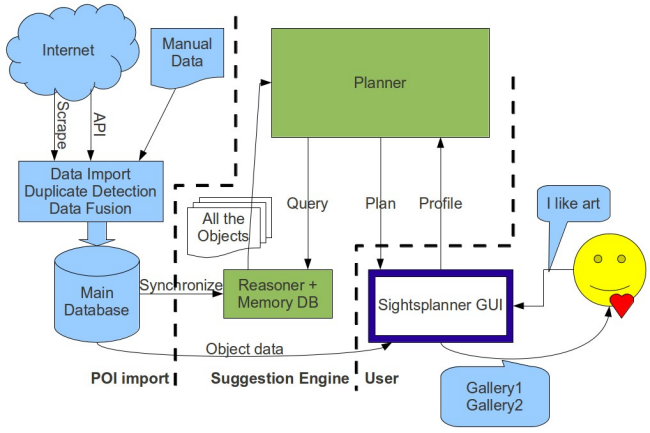

Fig. 2. System architecture

Object information is saved into the database using extended RDF triple structure (http://www.w3.org/TR/rdf-concepts/ , [2004]). Every fact about each object is conceptually represented as an object, property, value triple (in standard RDF terminology subject, predicate, object). We have extended this structure by adding new fields for every fact. The main additional fields are the unique triple identifier, connection score, source, and timestamp. The unique identifier is automatically assigned by the database engine. The connection score indicates the probability of correctness of the given fact. The score is within the range [0; 1] and indicates the intuitive likelihood that the fact is true. The source field stores the name of the source from where the fact is gathered (web portal name, manual insertion, generated etc.). The timestamp stores the date and the time when the fact was created.

The recommendation engine has to quickly handle a large amount of data: all properties of all objects and a relatively large ontology. Fetching all this data from the conventional relational database takes too much time. In addition to fetching the data we also would like to deduct new facts by applying rule-based reasoning. We have built a new reasoner-equipped fast in-memory database to serve our needs. This in-

memory database is implemented in shared memory as a library, without any continuously running processes.

Data imported from the web pages is synchronised from PostgreSQL to the memory database once a day. Currently the synchronisation works only one way: all the data from persistent database is copied to the memory database. We employ our specialized reasoner to derive new types, opening times etc.

The recommendation process verifies suitable objects by loading the tourism objects from the in-memory database and applying an object filter: objects that do not have the required properties (coordinates, opening/closing times), belong to a different locale or are not accessible during the given visit duration are ignored. For each loaded object the recommender assigns a ranking score based on a matching between object types and preferences taken from the user profile.

The recommender runs the planning algorithm that produces a number of candidate solutions. These are ranked using an aggregate score which is based on the average score of the objects included, as well as how the time is divided between different categories of objects, when compared to the relative importance of the categories in the user's profile. The top ranking solution is returned to the user. The output of the planner is a timetable of objects, containing a Unified Resource Identifier (URI), arrival and departure time and method of travel from the previous location for each object.

Using the URI-s from the planner output, the recommender retrieves additional data, including visualisation components, descriptions in the user's language and contact information from the PostgreSQL database and creates the timetable representation.

4 Data Discovery

For our recommender system we are currently scraping six different web portals with completely different structures. For every portal we have written mapping rules to normalise raw data for our system. Some of the objects in the recommender are based on dynamic, rapid changing information such as one-time events (concerts, performances etc.) which are scraped from the web daily. Another category of objects is formed by places of interest (POIs) with mostly static information - nothing changes in months or even years. The POI objects are updated typically once a month.

The importer can connect to several web pages using different manually created scraper algorithms. Each scraper downloads the content of the web pages, finds the necessary information (for example using Xpath (http://www.w3.org/TR/xpath/, [1999]) to extract title, description etc.), normalises data for our system and sends it to the Clusterer. The Clusterer compares the new object with the existing clusters by computing the similarity between the object and every cluster. From certain similarity value, the object is considered to belong to the cluster. In case of a match, the object

and the cluster data is merged. When the match is not found, the object creates a new cluster. More details about clustering and merging is presented in the next sections.

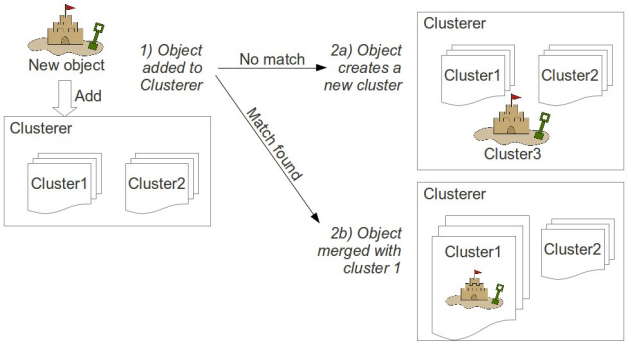

Fig. 3. Clustering of objects. An object is added to Clusterer. Case 2a illustrates the result when no matching cluster is found. Case 2b illustrates the result when a matching cluster is found and the obiect is added to that cluster.

The recommendation process uses properties *latitude*, *longitude*, *opening_time*, *type*, *popularity_tourist*, *visit_time* to calculate the recommendation for the user profile. The coordinates are mandatory and cannot be empty. If the opening times are missing, we use the default values based on the object type (restaurants and museums have different default opening times). We extract types from the *title* and the *description* values. *Tourist popularity* indicates the average amount of tourists visiting the object in one day. If the value is missing, it is defaulted to 1. *Visit time* represents the recommended visit duration for the given object. The default value is 30 minutes, which can be modified by the tourist.

Our system stores information in various languages. Properties like *name* and *description* have language code along with the textual value. With multilingual data we often have a problem of missing translations. Usually the web pages we are scraping only have data in one language which makes it harder to get all the information in all the languages. To overcome the problem of missing multi-lingual information, we use the Clusterer. An object is scraped from one web page which only has data in Estonian. We add the object into the database. If the system finds an object from another web site, which is in English, the found object is compared to the existing objects. Based on the location, types and other properties, the Clusterer knows that the Estonian object and the new English object describe the same object. The objects are added into the same cluster. Data from the objects is merged and the original Estonian object gets an additional English name and description. In addition to the multilingual textual information, the new scraped object can have additional metadata (phone number, homepage etc.) which can be merged to the existing cluster. If we cannot scrape information in other languages for an object, we run into the problem that some objects in our database do not have translations.

Table 1. An example of multilingual description

Object	Property	Value @ language	Score	Source
#church	#name	Oleviste Church @ en	1.0	#scraper1
#church	#name	Estonian name @ et	1.0	#scraper1
#church	#description	Translated English description @ en	0.8	#translator
#church	#description	Scraped English description @ en	1.0	#scraper2

We have implemented automatic translation functionality, which currently uses Google Translate API (http://code.google.com/apis/language/, 2011). We are translating title and description properties. The translated value is saved in the database with a lower probability score than for originally scraped values. In Table 1 we have presented a simple example with the language for every value and also the source for every fact. The scenario itself is as follows. We get the description of a church from one Estonian web page. The fact (description in Estonian) will get a score of 1.0. To be able to give some information about the object for foreign tourist, we translate the description automatically into English with a score of 0.8. At some point we will add a new web page scraper into our system which gets an English description for the same church. The scraped description would get a score of 1.0 and is therefore preferred over the automatically translated description (the fact with the source "#scraper2" is used to show English description). The scores for translation and different scrapers can be modified - instead of 1.0 and 0.8 other numbers can be used. For the web pages with low text quality, even the automatic translation can be preferred.

In the following subsections we will present more details about different data discovery topics. First we describe keyword extraction from object multi-lingual description. After that we will present our data clustering and merging technique with some examples.

4.1 Keyword extraction

The web pages usually do not provide enough meta-information about the object types or categories. Those are, however, crucial for suggesting tourism objects which match the user selected interests. Some web pages only offer a very general topic like "music". In our system we would also like to know the genre of the music, for example rock or classical. Therefore we have introduced a keyword extraction methodology in our system.

We realize conceptual indexing in an automatic fashion. Conceptual indexing usually involves firstly, the detection of relevant concepts, and secondly, the calculation of a relevance weight for each detected concept. We follow the approach of projecting the ontology onto the document by extracting all the concepts the following way:

As mentioned above, we store object information in different languages. To get object types, we analyse all the description properties in all the languages. Every language

has its own keywords which match with possible object types. The descriptions are parsed and all the matched keywords are found. The keywords are grouped by corresponding object types. The object will get all the found types with certain scores. The type which corresponds to more extracted keywords, gets a higher score and vice versa.

4.2 Data clustering

We have already given a brief overview of clustering. In this subsection we will present more details about the process. First, we have to define the cluster. When data is scraped from the web into our system, everything is stored as an imported object (referred as "object" or "imported object"). Cluster itself is also a tourism object (referred as "cluster", but sometimes also as "object"). It represents one physical object or event. If the same church is defined in different web pages resulting, for example, three imported objects, the Clusterer should put all the imported objects into one cluster. Cluster can consist of one or more imported objects, it has all the information (facts) imported object has. In fact, the only main difference from an imported object is that cluster has an additional fact which just states that the given object is cluster. During Clusterer work, imported objects are left unchanged. Instead, clusters will be modified or added when needed. The recommendation process uses only cluster objects. After the import and clustering every imported objects should be in a cluster. If no similar objects are found, then the number of clusters will be the same as the number of imported objects.

Figure 3 describes how the Clusterer works. A new object (imported object) is sent to the Clusterer. The Clusterer compares the given object to every cluster. If no matching cluster is found, a new cluster is created and the object is added to the cluster. If a matching cluster is found, the object will be added to the matching cluster. Facts about the object and the found cluster are merged. We will describe data merging in subsection 4.4.

Comparing an object and a cluster is done by finding a similarity value between the two. Similarity in our system is a number in range [0.0; 1.0]. If the similarity is above a certain threshold, two objects are considered to be the same. More about the similarity calculation will be presented in the next subsection.

Comparing objects for similarity is expensive in terms of time consumption. To avoid comparing an object to every cluster, we try to find a smaller number of candidates for a new object. We have implemented location based candidate search: there is no point in comparing objects which are too far away from each other. Currently, with 3000 clusters and 300 new objects, the total running time of comparison process is around 10 minutes on a quad-core laptop. The number depends on the concrete data and the number of similar objects. If we had 5000 objects, it would probably take hours. The clustering process is usually run once a day. As mentioned, we regularly update only event data. More static POI data is updated only once a month. To improve the speed of clustering, we are looking into better candidate selection to get fewer candidates, different clustering algorithms and simpler comparison functions.

4.3 Similarity

Different data sources can describe the same physical tourism object or event. Usually those sources do not link data together. When importing the information, we end up with several different objects in our system. Now it is up to us to decide, which of those objects are actually the same. We are using the *similarity* to decide that. The system calculates *similarity value* of two objects and if the value is above a certain threshold, the objects are considered to be the same and are put into the same cluster.

We have implemented several functions named *comparators* which deal only with certain characteristics. For example, we have a separate function for getting location similarity (whether two objects are similar based on the coordinates or address) or for titles (whether the titles of the objects are similar) and so on. Every function gives a result in range [0.0; 1.0]. All the *comparators* have *importance value*, which states how important the result of this function is in the total similarity. The calculation of *total similarity* can be viewed as a weighted mean. In our implementation, the formula is:

$$similarity = \frac{2 * location + title + type + properties + event\ start\ time}{6}$$

All five elements of the sum are different similarity functions. As can be seen, the *importance* for *location* is twice as high as for other elements. Importance values are currently set manually. We plan to use machine learning algorithms to learn better values. If the *similarity* is above 0.9, the system puts the objects into the same cluster.

In the next subsections we will describe the main *comparators*. Every *comparator* provides a *check function* which is used to test whether the corresponding comparator can be applied. For example, if one of the objects is not an event, *start time comparator* cannot be used as the similarity result would be 0. When the *comparator* is not used, it is not included in *total similarity* calculation. In the case of non-events, the *similarity* formula will have 5 as divisor (instead of 6).

4.3.1 Location comparator

One of the most important comparisons in our system uses object distances. Objects in database are described by *latitude* and *longitude*. Using coordinates, we calculate direct distance between the objects. We have defined three ranges: first is allowed distance without penalty, second one is allowed distance with penalty, and third one is distance not allowed. Currently, if the distance is smaller than 10 meters, the *location similarity* will be 1.0. If the distance is between 10 and 100 meters, the *similarity* will fall linearly: 10 meters would still yield in *similarity* of 1.0, where 100 meters would give 0.0. Distances over 100 meters would give 0.0 as a result.

Comparing only the coordinates might be misleading. As we scrape information from various sources, they can apply different coordinate systems (Google versus local Estonian system) or simply provide different location for the same object. For

example, if the building is 200 meters long, one source can point to one side of the building as the other can point to the opposite side. If we compare the locations, we would get similarity 0.0. To overcome the problem, we also compare addresses.

Address comparison is done in two different ways. The first one is string comparison: we remove common words like “street”, “road” etc., extract the street name and the building number. If both the street part and the building part match, the similarity will be 1.0. We also use Google maps API (http://code.google.com/apis/maps/, 2011) to get coordinates for the address. Google accepts address in various formats and returns one certain location for different address strings. We use coordinates provided by Google and compare locations as we did with our original coordinates.

4.3.2 Event start time comparator

Comparing only the location does not help us in the case when some events take place at the same building. Therefore we compare starting time of the events. Similarly to location comparison we have defined three ranges. If the time difference is below 15, the *similarity* will be 1.0. The difference between 15 and 60 minutes would decrease the *similarity* from 1.0 to 0.0 linearly.

4.3.3 Text comparator

With only *location* and *start time comparators* we would still experience problems, for example a cinema with three movies starting at the same time. The only way to make difference is to compare titles. Another example would be small shops or cafés inside a huge supermarket. Again, we need to compare titles.

As with location, we have implemented several different comparison functions. First is to compare all the words from objects' titles. We try to remove common words like “café”, “shop” to find out that “Cafè X” and “Cafè Y” are not similar although “cafè” matches. With different languages, it is often a problem to know the common words.

Another example is “McDonalds” and “McDonald's”. Depending on how the splitting into separate words is done, the problem still occurs when there are some symbols different (due to misspelling or special symbols). For the given case, we have introduced two string comparison algorithms: Levenshtein distance (also known as edit distance) and trigram comparison. They are both good in the case of misspelled symbols or when only small number of symbols are different. For example “Mc Donald's” and “McDonalds” are similar when using those methods.

Described algorithms do not solve all the cases. For example, in a shopping centre X there are two eating places: “McDonald's X” and “Pizzeria X”. The address is the same, they both are eating places and X part of the word is matching. The *title similarity* will not be 1.0, but it is high enough to push the *total similarity* above the threshold. Here we have an idea to use the knowledge, that X is a shopping centre (if it is present in our database) and consider this part of the titles as common word. Then we would compare “McDonald's” and “Pizzeria”, which would yield in similarity 0.

There are several cases where we still have to find a better solution. Often the title comparison is decisive whether the objects are the same or not. This is one priority in our current work.

4.4 Data merging

After two objects have been put into one cluster, the cluster has to have representative information. An easy solution would be to just add all the facts from every imported object to the cluster. This may end up with objects which have two different names. To simplify the task of presenting data with user interface, we have implemented a merging process, which tries to remove obsolete information about the object.

During the clustering process all the facts from imported objects are added to the clusters. The merger component is run separately after clustering. Current implementation keeps the value with highest connection score for every property. Certain properties, for example all the imported types, are always kept.

Often values have the same probability score. For that case we have introduced a source priority list: value from the source which has a higher priority is preferred. Current approach has weaknesses and more sophisticated selection based on the context should be implemented.

5 Experiments and Results

On Sightsplanner.com we currently have about 2200 scraped objects about Tallinn: 1200 POIs and 1000 events. The number of different events is about 150, but we keep every occurrence as a separate object (a play which takes place three times will be stored as three objects in our database). We scrape data from six different portals both in English and in Estonian. Every night we automatically update event information.

We have gathered a dataset of Tallinn eating places to measure the clustering precision. We have a total of 1700 eating places (including duplicates). Total number of different objects when duplicates are merged is 800. Eating places are gathered from 5 different sources, which results in high number of comparison space (the number of possible object pairs to compare is above 10 000 000). Some objects have to be merged from 5 different data sources, where others only from 2 sources. Given the size of comparison space, we have gathered a different dataset to measure the clustering accuracy.

We have manually found all the duplicates in a dataset of about 3500 scraped objects for another city Riga. Created dataset along with all the similar objects is used as a control dataset. We have used the same configuration of the Clusterer as with Tallinn data to get the results. In the given dataset, there are 77 duplicate objects, e.g. 154 objects will create 77 clusters, 2 objects in every cluster. The results with our current implementation are 87% precision (67 objects found from 77) and 87% recall (10 false negatives). 90% precision should be achievable, the main problem is title

comparison, where we would to take into account additional context information like object type, location, common keywords and existing shopping centre names etc.

Our next goal is to apply the same accuracy measure on Tallinn data. We have to create a subset of our data to limit the search space in order to be able to manually indicate all the duplicates. The selected subset can then be used as a training data for machine learning, which could optimize the similarity function parameters to improve clustering and merging of the objects.

6 Related Work

Geographical information extraction task is covered by Alves et al. (2009), where they describe KUSCO system which searches internet to enrich a Place of Interest (POI) data. Words from the results are used to discover the *Semantics of Place*. In the work they use natural language processing to extract nouns which will be used as concepts for entities. We are currently extracting keywords only from specified sources and are looking into extending the lookup to the whole web, which would give us more information and also help to improve deduplication process.

In (Tré, 2010) a formal theory about POI deduplication and data merging is given. The article focuses mainly on *coreferent* (*clustered* in our terminology) POI data merging. The approach described in the paper is somewhat similar to our methodology. We do not calculate true and false probabilities. Instead we make use of the probabilities of our facts. In some cases (for example for descriptions) proposed techniques could be used to combine descriptions. As mentioned in the article, they do not deal with multilingual data. In one example, a translated title for a POI is lost due to merging. Our system has a requirement to merge data separately for every language.

Bleiholder and Naumann (2008) have given a good overview of data merging techniques. The describe different approaches for different requirements. Along with every approach, they have provided SQL examples. When using terminology from the article, we are using mostly *Take the information* and *Trust your friends* strategies. In addition we use fact probabilities to choose the one with the highest score.

A very similar approach is used in (Zheng, 2010) where the authors present a problem of finding near duplicate records in location datasets. They compare POI titles, addresses and types. As they have proposed a structure for the address field, they probably have rather high distances between objects. In our case, most of the objects are located in one part of the city. Our lowest hierarchy should be street, which is actually included in our address similarity calculation. Our current similarity measures are also near 0.9, which probably improve after we apply machine learning.

7 Conclusions and Future Work

In this paper we have described the tourism recommender Sightsplanner. A brief overview of the system architecture is given. The main focus of the paper is on the data discovery topics, where data importing, entity disambiguation and data fusion processes are described. We have presented the main functionality about different data processes along with some problems we have to overcome in order to improve the quality of the system and data itself.

In our current implementation we are using six different data sources. The goal is to be able to use an unlimited number of portals from the web. To achieve this, we have to improve the clustering process which would enable gathering information which for example, does not have coordinates. If we have an object in our database, we could just look for a more detailed description. Currently, the disambiguation depends strongly on location. We could also add information from user generated content by the title of the object: a tourist has visited one restaurant and writes about that in a blog. If we can identify the object in our database, we can add the new data into our cluster and merge all the facts.

We are already working on adding open linked data into our importer. There are several web portals which could be used to retrieve linked data (DBpedia, OpenStreetMap etc.). Another future improvement will be user feedback on both data and recommendations. Based on that we could evaluate the quality of our planner and information retrieval.

References

Alves, A., Pereira, F., Biderman, A. & Ratti, C. (2009). Place Enrichment by Mining the Web. In M. Tscheligi, B. de Ruyter et al. (Eds.), *Ambient Intelligence,* 5859: 66-77. Springer Berlin.

Bleiholder, J. & Naumann, F. (2008). Data Fusion. *ACM Computer Surveys*. 41(1): 1–41.

Luberg, A., Tammet, T. & Järv, P. (2011). Smart City: A Rule-based Tourist Recommendation. *Information and Communication Technologies in Tourism 2011:* 51-63. Springer New York.

Tré, G. D. & Bronselaer, A. (2010). Consistently handling geographical user data: Merging of coreferent POIs. *Fuzzy Information Processing Society (NAFIPS), 2010 Annual Meeting of the North American,* 1(1): 117-122. IEEE New York.

Zheng, Y., Fen, X., Xie, X. et al. (2010). Detecting nearly duplicated records in location datasets. *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems* 1(1): 135-143. ACM New York.

Acknowledgements

This work has been funded by the European Commission as part of the TEAM IAPP project (grant no. 251514) within the FP7 People Programme (Marie Curie).

Appendix 5

V

A. Luberg, P. Järv, K. Schoefegger, and T. Tammet. Context-aware and multilingual information extraction for a tourist recommender system. In *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies*, i-KNOW '11, New York, NY, USA, 2011. Association for Computing Machinery

Context-aware and Multilingual Information Extraction for a Tourist Recommender System

Ago Luberg
Eliko Competence Center
Teaduspargi 6/2, 12618
Tallinn, Estonia
Tallinn University of Technology
Ehitajate tee 5, 19086
Tallinn, Estonia
ago.luberg@eliko.ee

Karin Schoefegger
Graz University of Technology,
Knowledge Management Institute
Inffeldgasse 21a
8010 Graz Austria
k.schoefegger@tugraz.at

Priit Järv
Eliko Competence Center
Teaduspargi 6/2, 12618
Tallinn, Estonia
Tallinn University of Technology
Ehitajate tee 5, 19086
Tallinn, Estonia
priit@cc.ttu.ee

Tanel Tammet
Eliko Competence Center
Teaduspargi 6/2, 12618
Tallinn, Estonia
Tallinn University of Technology
Ehitajate tee 5, 19086
Tallinn, Estonia
tammet@staff.ttu.ee

ABSTRACT

We present information extraction for a semantic personalised tourist recommender system. The main challenges in this setting are that information is spread across various information sources, it is usually stored in proprietary formats and is available in different languages in varying degrees of accuracy. We address the mentioned challenges and describe our realization and ideas how to deal with each of them. In our paper we describe scraping and extracting keywords from different web portals with different languages, how we deal with missing multi-lingual data, and how we identify the same objects from different sources.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Content Analysis and Indexing

General Terms

Theory

Keywords

Multilingual information retrieval, keyword extraction, object similarity, recommender system, tourism

1. INTRODUCTION

In this paper we describe information retrieval and extraction for a semantic tourist recommender system Sightplanner. We are using semi-automatic scraping of different web portals. The data is normalized into our ontology. The main challenges in this setting are that information is spread across various information sources, it is usually stored in proprietary formats and is available in different languages in varying degrees of accuracy (e.g. an Estonian museum might provide a rich description of events in Estonian but only a short introduction in English).

In the following two sections we will provide a short overview of the overall system and its technical parts. In section 4 we describe the data gathering process in more detail. We end our paper with an overview of related work and conclusion.

2. TOURISM RECOMMENDER SYSTEM

Sightsplanner is a semantic recommender and route composer system for tourists. A tourist can specify her location, time and duration of the visit and her preferences about different types of objects and events. Based on the created tourist profile, the recommender identifies interesting objects for the given user. For each found object a ranking score is found. The objects that the user probably likes have a higher score and vice versa. A planning mechanism organizes the objects and events into a trip timetable. In an interactive feedback cycle, the tourist has the option to modify the suggested trip. To calculate the final list of recommended objects, the following processes are involved: (a) Object verification process, (b) Matching process, (c) Planning process, (d) Result representation process, and (e) Feedback process.

Figure 1: GUI first page

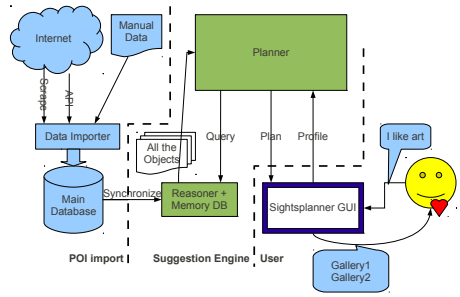

Figure 2: System architecture

The recommendation process starts when the user opens the web page and defines her interests by using a slider-based approach for adjusting individual preferences. In figure 1 the first page of the user interface is shown. The tourist can indicate her interest in seven main topics moving the corresponding sliders. Each of them also has subtopics. In the figure the "Events" slider is "opened" and its subsliders are shown. The tourist has stated that she likes "Museums and arts" and "Architecture and city" very much. She also likes some "Eating out", "Shopping" and "Events". Especially she likes "Jazz" and "Rock concerts". Every topic in the user profile matches a type in POI properties. The user can also specify the start date and time of her visit, visit duration and preferred travelling method.

After the tourist has selected her interests, she starts the overall recommendation process. The user profile is sent to the planner, which is responsible of returning a personalised recommendation for the given profile. The planner uses all the relevant data from the memory database. All the objects in the requested city which also are opened during user's visit will be processed. Based on the object types, location, opening time and some other properties, different trip timetables are created for the user. The planner has a time limit, after what it has to return the best trip it has found so far - this is then presented to the user as a timetable and as a map.

The recommended plan can be modified by the user starting an interactive feedback cycle: she can remove some objects or change visit time for objects and re-run the recommendation based on the performed changes: after the modification, the planner takes the changes into account and a new recommendation is created for the user.

The PostgreSQL database [12] of Sightsplanner contains all the known facts about the tourism objects. The data needed for computing the recommendation is periodically propagated to the in-memory database. The PostgreSQL database itself contains additional properties that are used for displaying the recommendation to the user.

3. TECHNICAL OVERVIEW

The main components of the system are presented in figure 2. In this section we will give a brief overview of the different parts. First we will look into data gathering part. Then we will describe the whole process with some details about each of the components.

Table 1: A fragment of POI data in database

Object	Property	Value	Score
#town_hall_sq	#name	Town hall square	1.0
#town_hall_sq	#type	#architecture	0.9
#town_hall_sq	#type	#city	0.9
#town_hall_sq	#type	#shopping	0.4
#town_hall_sq	#latitude	59.437165314024	1.0
#town_hall_sq	#longitude	24.745137236142	1.0

Information about the objects stored in our database is spread across various information sources, it is stored in different proprietary formats and is available in different languages in varying degrees of accuracy. The more detailed and coherent we can gather this data, the better recommendations and scheduling we can provide to the user. One crucial requirement is for example the POI (point-of-interest) data.

In our current implementation we scrape information about objects in a semi-automatic fashion. First, we crawl different web portals and normalize the gathered data into our ontology. In some cases, the information available in the provided portals is not complete. Therefore we can also add data manually: we can add a new complete object or modify some properties for the existing objects. All this information is stored in our main database which in our system is PostgreSQL [12]. In section 4 we will describe the data importer component and its subcomponents more deeply.

POI information is saved into database used extended RDF triple structure [13]. In our database, every detail can be represented with a triple: object, property, value (originally subject, predicate and object accordingly). We have extended this structure by adding fields like unique identifier, connection score, source, timestamp. Unique identifier is automatically assigned by the database engine. Connection score indicates the probability of correctness of a certain fact. The score is within the range $[0; 1]$. If the score is 1.0, then the fact is likely to be true. The score 0.0 states that

the fact is unlikely to be true. The source stores the name where the fact is gathered (web page name, manual insertion etc.). The timestamp stores the date and time when the fact is created. A small fragment of the database is presented in table 1. For the example to be more clear, we have only shown object, property, value and score fields.

The recommendation engine has to quickly handle a large amount of data: all properties of all POIs and a relatively large ontology. Fetching all this data from the conventional relational database takes too much time. In addition to fetching the data we also would like to deduct new facts by applying rule-based reasoning. In particular, it is not suitable to run the reasoning engine by querying all the potential premises of rules from the conventional database for each rule application during derivation. The reasoning engines are typically built on specialised efficient main-memory data stores.

We have built a new reasoner-equipped fast in-memory database to serve our needs. This in-memory database is implemented in shared memory as a library, without any continuously running processes. Each update or query to the database is performed as a call to the corresponding function. In order to enable parallel queries, we have implemented a fast global read/write lock using spinlocks. Those prohibit simultaneous data addition and reading from the global data area. Each query process has an additional local non-shared data area for query input and temporary results. Thus it is possible to conduct several derivation queries in parallel, avoiding the need to copy common data to all the separate simultaneous derivation processes.

All the imported data from the web pages is stored in PostgreSQL. In order to make the same data available for memory database, we synchronize the data once a day. Currently the synchronization works only one way: all the data from persistent database is copied to the memory database. The employed reasoner is used to derive new types, opening times etc. For example, we have a rule

$$fact(?X\ type\ architecture\ 0.9{*}?N)\ :- \quad (1)$$
$$fact(?X\ type\ church\ ?N) \quad (2)$$

which says that if the object X has a type "church" with score N, then the same object would get a type "architecture" with score $0.9 * N$.

The recommendation process verifies suitable objects by loading the tourism objects from the in-memory database and applying an object filter: objects that do not have required properties (coordinates, opening/closing times), belong to a different locale or are not accessible during the given visit duration are ignored. For each loaded object the recommender assigns a ranking score based on a matching between "#type" properties of the object and the preferences taken from the user profile.

The array of objects is sorted in descending order of the calculated ranking scores. The recommender then runs the planning algorithm that arranges the objects in a timetable for the user while attempting to maximise the individual object rankings that are being included in the recommendation. There are several constraints to the timetable that need to be satisfied - the tourism objects have opening times (therefore, the optimization problem as modelled in Sightsplanner may be classified as an instance of OPTW [18] problem) and each of the preferred categories that was present in the user profile should be represented (due to the latter requirement the OPTW problem in Sightsplanner could be classified as multi-objective optimization, however we have chosen to aggregate this criteria into the overall score of the recommended timetable).

The planning problem is solved using a meta-heuristic search based on GRASP [15]. The method of search was originally described by Vansteenwegen et al, who employed this type of heuristic in their Citytripplanner recommender [17] . Sightsplanner uses a similar approach, where the candidate objects chosen in GRASP iterations are added incrementally into the current solution, therefore having no distinct local search phase. The balance criteria that requires objects from varied categories to be present in candidate solutions is used to additionally guide the selection of items in the GRASP search.

When computing the solution to the maximisation problem, the "#latitude" and "#longitude" properties of the objects are used to estimate the time required to move from one candidate object to another. The "#opening_time" properties are used to determine, whether an object is accessible at any given moment.

The meta-heuristic search produces a number of candidate solutions which are ranked using an aggregate score which is based on the average score of the objects included and the balance criteria. The top ranking solution is returned to the user. The output of the planner is a timetable of objects, containing a Unified Resource Identifier (URI), arrival and departure time and method of travel from the previous location for each object.

Using the URI-s from the planner output, the recommender retrieves additional data, including visualisation components, descriptions in the user's language and contact information from the PostreSQL database and creates the timetable representation.

4. DATA DISCOVERY

For our recommender system we are currently scraping six different web portals with completely different structures. Raw data obtained from the portals is then normalized for our system. Some of the objects in the recommender are based on dynamic, rapid changing information such as one-time events (concerts, performances etc.) which are scraped from the web daily. Another category of objects is formed by places of interest (POI-s) with mostly static information - nothing changes in months or even years. The POI objects are updated typically once a month.

A brief overview of the architecture of our importer is presented in the figure 3. The importer can connect to several web pages using different manually created scraper algorithms. Each scraper downloads the content of the web

Table 2: A small fragment of profile ontology

Parent	URI	Description
Profile	#events_and_happenings	Events and happenings
#events_and_happenings	#events_and_happenings_concert_classic	Classical music concert
#events_and_happenings	#events_and_happenings_concert_rock	Rock concert
#events_and_happenings	#events_and_happenings_concert_jazz	Jazz concert

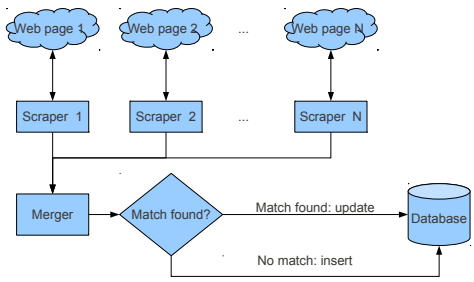

Figure 3: Data scraping

pages, finds the necessary information (for example using XPath[20] to extract title, description etc), normalizes data for our system and sends it to the Merger. The Merger compares the new object with the existing ones by computing the similarity between the objects. From certain similarity value, the objects are considered to be the same. In the case of match, the existing object in the database is updated with new scraped data. If there is no matching object in the database, the scraped object is added. More details about the merging is presented in the next sections.

The recommendation process uses properties "#latitude", "#longitude", "#opening_time", "#type", "#popularity_tourist", "#visit_time" to calculate the recommendation for the user profile. The coordinates are mandatory and cannot be empty. If opening times are missing, we use default values based on the object type (restaurants and museums have different default opening times). We extract types from the title and the description values. Tourist popularity indicates the average amount of tourists visiting the object in one day. If the value is missing, it is defaulted to 1. Visit time represents the recommended visit duration for the given object. The default value is 30 minutes. The tourist can modify the suggested visit duration for every object. Along with mentioned properties, there are many other meta fields.

Our system stores information in various languages. Properties like "#name" and "#description" have language information along with the textual value. With multilingual data we often have a problem of missing translations. Usually the web pages we are scraping only have data in one language which makes it harder to get all the information in all the languages. To overcome the problem of missing multi-lingual information, we use the Merger. An object is scraped from one web page which only has data in Estonian. We add the object into the database. If the system finds an object from another web site, which is in English, the found object is compared to the existing objects. Based on the location, types and other properties, the Merger knows that the Estonian object and the new English object describe the same object. The Merger updates the original Estonian object and adds the English name and description. In addition to the multilingual textual information, the new scraped object can have additional metadata (phone number, homepage etc.) which can be added to the existing object. If we cannot scrape information in other languages for an object, we run into the problem that some objects in our database do not have translations because we could identify a similar object by scraping different portals but the majority of objects is still not available in the other languages necessary.

We have implemented automatic translation functionality, which currently uses Google Translate API [10]. We are translating title and description properties. The translated value is saved in the database with lower probability score. When the text is scraped from a web page, then the score is higher than for the automatically translated text. In table 3 we have presented a simple example. In the table we have shown the language for every value and also the source for every fact. The scenario itself is as follows. We get the description of a church from one Estonian web page. The fact (description in Estonian) will get a score 1.0. To be able to give some information about the object for foreign tourist, we translate the description automatically into English with a score 0.8. At some point we will add a new web page scraper into our system which gets English description for the same church. The scraped description would get a score 1.0 and is therefore preferred over the automatically translated description (the fact with the source "#scraper2" is used to show English description). The scores for translation and different scrapers can be modified - instead of 1.0 and 0.8 other numbers can be used. For the web pages with low text quality, even the automatic translation can be preferred.

In the following subsections we will present more details about different data discovery topics. First, we describe keyword extraction from object multi-lingual description. After that, we present our data merging technique with some examples.

4.1 Keyword Extraction

The web pages usually do not provide enough meta-information about the object types or categories. For our recommender system those are very important to suggest objects which match with user selected interests. The web page can only offer a very general category like music. In our system we would also like to know the genre of the music, for example rock or classical. Therefore we introduce a keyword extraction methodology in our system.

Table 3: An example of multilingual description

Object	Property	Value@language	Score	Source
#church	#name	Oleviste Church@et	1.0	#scraper1
#church	#name	Estonian description@et	1.0	#scraper1
#church	#description	Automatically translated description@en	0.8	#translator
#church	#description	English description@en	1.0	#scraper2

Table 4: An example of keywords to extract from text

Keyword	Ontology term / type	Score
rock	#concert_rock	1.0
pop	#concert_pop	1.0
musical	#concert_pop	0.8
jazz	#concert_jazz	1.0
music	#events_and_happenings	0.7
folk	#events_and_happenings	0.8

We realize conceptual indexing in an automatic fashion. Conceptual indexing usually involves firstly, the detection of relevant concepts, and secondly, the calculation of a relevance weight for each detected concept. In concept detection from documents, two alternative ways can be distinguished: The first projects the ontology onto the document by extracting all multi-word concepts from the ontology and then identifies those occurring in the document. The second approach follows the opposite way, it projects the document onto the ontology by combining adjacent words in text phrases. Usually, the words in their original form are looked up in the the ontology and if necessary, their base forms are used to resolve the problem of word forms. For multi-word concepts, the longest term for which a concept is detected, is used. We follow the first approach in the following way:

As mentioned above, we store object information in different languages. To get object types, we analyse all the description properties in all the languages. Every language has its own keywords which match with possible object types. The descriptions are parsed and all the matched keywords are found. The keywords are grouped by corresponding object types. The object will get all the found types with certain scores. The type which corresponds to more extracted keywords, gets a higher score and vice versa.

In the table 4 we have presented a small example of our keyword base. We have shortened the ontology terms for the first four rows (each should have "#events_and_happenings_" as a prefix). The score indicates the confidence of the given match. For example, if the description consists of words "rock" and "musical", the event would get corresponding types with scores 1.0 and 0.8 accordingly. If the same description would consist a word "pop", then "#events_and_happenings_concert_pop" type would get highest score amongst matching keywords, which is 1.0. If the description consists more of the same keyword, the score is multiplied with a factor, which currently in our system is 2.0.

4.2 Data Merging

The POI data comes from different sources and often many web pages can have information about the same object. If all the data sources would use semantic web technology, the same object would have the same URI, which would make scraping easier. In our case we do not have any information about the similarity of objects in different sources. Therefore we have implemented our own merger, which was briefly mentioned in section 4.

We have a very simple general algorithm for calculating object similarity which is presented in algorithm 1. We have defined several different comparators, which all have a comparison function. Each function compares certain properties or values and returns a value in range $[0, 1]$. Along with the function, comparators have *importance* value. The higher the importance value is, the more the given function affects the total similarity. The importance is calculated as a ratio to total importance. For example, if we have two comparators: one has importance 2 and the other has importance 3. The total importance would be 5. Now the first comparator would affect the total similarity $2/5 = 40\%$, the second one $3/5 = 60\%$. The first similarity function can compare titles. If the titles match, it returns 1 and the total similarity would be $0.4 * 1 = 0.4$. The second function compares locations. The objects are not located in the same place, so the outcome of the function is 0.5, which would add to the total similarity $0.6 * 0.5 = 0.3$. And the total similarity would be 0.7.

In the algorithm we also have used *check_function* which returns boolean whether the function can be applied for the given objects. For example, there is a comparison for events which compares start times. There is no point of employing the same comparison to POIs, as they do not have start times (they do have starting opening time). If the check function returns false, then the given comparator is left out from both *total importance* and *total similarity* calculations.

For every scraped object, the best match is found from the existing objects. If the similarity is above certain threshold, the objects are considered to represent the same object. In our current implementation the threshold for matching objects is 0.9.

We have compared the Merger results manually for Tallinn eating places. We used two different web portals to scrape the data: http://tourism.tallinn.ee with 270 object and http://toidukohad.eu with 510 object. Our current configuration for comparators and the threshold resulted in 90% of accuracy on deciding the match. One of the main problems was matching eating places inside a supermarket. Also, common names like "cafe" and "pub" matched some objects,

Algorithm 1 General similarity calculation

procedure COMPARE($object1$, $obect2$)
  $Comparators \leftarrow all\_the\_comparators$
  $total\_importance \leftarrow 0$
  for all $comp \in Comparators$ **do**
    if $not\_defined(comp_{check\_function}) \vee comp_{check\_function}(object1, object2)$ **then**
      ▷ check function not defined or returned true, then this comparator is used
      $total\_importance \leftarrow total\_importance + comp_{importance}$
    end if
  end for
  $total\_similarity \leftarrow 0$
  for all $comp \in Comparators$ **do**
    if $not\_defined(comp_{check\_function}) \vee comp_{check\_function}(object1, object2)$ **then**
      $similarity \leftarrow comp_{comparison\_function}(object1, object2)$
      $max\_similarity \leftarrow comp_{importance}/total\_importance$
      $total\_similarity \leftarrow total\_similarity + similarity * max\_similarity$
    end if
  end for
  return $total\_similarity$
end procedure

which actually were different.

This field needs some more testing and experimenting. We already have started with few different cities to test our Merger. The gathered test data will certainly give us more information on how to tune our configuration setup. We can also get more accurate evaluation results with different datasets.

Next we will present some comparators we are successfully using in our system. Note that each comparator provides a *check function* which is used to test whether the corresponding comparator can be applied. E.g. for the Start time comparator if one of the objects is not an event, the check function would return false and the given comparator is not used for the given objects as non-event objects do not have fixed starting time and the comparison would always yield to 0 result.

4.2.1 Location comparator

One of the most important comparisons in our system calculates object distances. Every object in database is described by latitude and longitude properties. If we compare objects, we calculate direct distance between the two objects. Along with the comparator we have defined three ranges: first range is allowed distance without penalty, the second one is allowed distance with penalty and the third one is not allowed distance. If the distance between the objects is less than 10 meters, the similarity based on the location is 1.0. If the distance is between 10 and 100 meters, then the similarity would fall linearly. 10 meters would still yield for similarity 1.0, 100 meters would give 0.0. Distances over 100 meters would give 0.0.

Comparing only the coordinates itself might be misleading. As we scrape the locations from various sources, they can apply different coordinate systems (Google [9] *versus* local Estonian system [14]) or simply provide a different location for the same object. For example, if the building itself is 200 meters long, one source can point to one side of the building as the other can point to the opposite side. If we compare the locations, the function would return 0.0. To overcome this problem, we also compare addresses.

Address comparison is done in two different ways. The first one is to compare strings. Before comparing we remove common words like "street", "st", "road", "avenue" etc. from the addresses. We also extract the building and the apartment number. Then we compare the street part and building part separately. If we find that both of them match, then we return 1.0 as location similarity. If we do not find a complete match, we use Google maps API [8] to get the coordinates for the addresses. Google accepts addresses in different formats and returns one certain location for the same address. We compare coordinates provided by the Google and calculate the similarity as we did with our original coordinates.

4.2.2 Event start time comparator

The location comparison on its own often does not satisfy our needs. Especially if there are events taking place in the same location. Therefore we also need to compare the starting times of the events. As with the distance we use here different time ranges. If the difference between the events is less than 15 minutes, the time based match return 1.0. From 15 minutes to 60 minutes, the similarity falls linearly to 0.0. Events with start time different more than one hour would get similarity 0.0.

4.2.3 Text comparator

With location and start time comparators, we still experience problems with some objects. For example a cinema with three movies taking place at the same time at the same address, only the room is different. Another example would be shops and cafes in supermarkets. As those are not events, start times are not compared. And as they are located in the same building, the address and the location are the same. Therefore we need another comparison for this kind of objects. We have introduced a text comparator.

We divide the text into keywords and find the matches in the other object's texts. We remove some common keywords like

"Cafe", "shop" etc. Otherwise the titles "Cafe X" and "Cafe Y" already would have some similarity, but actually they are not similar (in the context of being the same place). Another problem arises with supermarket name. If the supermarket is named X, then there can be "Hesburger X" and "McDonald's X". Again, X would match in both names and therefore would be similar. We have tried to remove full object names from titles, but it only removes a small number of problems. Often the object itself has a name "supermarket X". Now we would like to know that X is so called "common" part of the name and matching X does not say anything about the similarity of the objects.

Another problem comes from translations. For the objects, which do not have information available in all the needed languages, automatic translation is used. And when comparing automatically translated name with a professionally translated name, they often do not match.

In our system, the importance of the text comparison is lower than used for location and start time comparators.

4.2.4 Examples

To illustrate the comparison process and results of the calculations, we provide two examples.

1) We have two objects:

Table 5: An example 1 of comparing objects

Property	Object A	Object B
#title	McDonald's Rocca	McDonald's Rocca
#latitude	59.4258329	59.4258492
#longitude	24.65024	24.65464
#address	Paldiski mnt 102	Paldiski mnt. 102

When we compare locations by coordinates, we get 248 meters as the distance between the objects. Comparing addresses says it is the same building, which means that the location similarity is 1.0. If we had just used the coordinates-based calculation, the similarity would have been 0.0. The start time comparator will be skipped because it has *check function* which checks whether both objects are events (given objects are not events). The similarity between titles is also 1.0. Therefore the Merger gets total similarity 1.0, which means the objects represent the same physical object.

2) We have two objects:

Table 6: An example 2 of comparing objects

Property	Object A mistoimub	Object B
#title	The Little Prince	Carmen
#latitude	59.4346093673	59.4341302
#longitude	24.7507912449	24.7506081
#address	Estonia pst. 4	Estonia pst 4
#start_time	19:00	19:00

In this example there are two events at the same place. Both events take place in Estonian National Opera in the same time, but in different rooms. First, if we compare coordinates, we get the distance 54 meters. This would give the location similarity 0.5 (100 meters would be 0.0, 0-10 meters would be 1.0). Again, we need to compare addresses (which are the same) and get the actual similarity 1.0. Comparing start times we get the similarity 1.0. The only comparable difference presented in our example is the title. As there are no matching words, the title similarity will be 0.0. We have selected the title comparator importance so that in the current situation it would affect the total similarity enough to fall below our selected matching threshold. The real numbers in our implementation are: threshold 0.9, title comparator importance 0.15 (with 0.0 title similarity, the maximum total similarity can be 0.85, which would be lower than the threshold). In addition to mentioned comparators, we compare event types (ballet *versus* opera), event homepage etc. The total similarity between those objects is actually 0.7.

5. RELATED WORK

Early approaches for creating fully automatic semantic annotations are for example implemented in MnM [19], a tool for ontology-driven automatic and semi-automatic annotation of web pages or KIM [11] is another tool for automatic semantic annotation of web pages based on light-weight upper level ontologies. The whole research area of 'information extraction' is dedicated to this problem (e.g. [6]). They either extract metadata from the web site's underlying databases, learning algorithms based on textual content of web pages and/or NLP techniques. Most of these methods use a pre-defined ontology as the semantic model for the annotations. None of these approaches implement methods how to deal with missing or ambiguous information. Information extraction in a multilingual scenario is for example realized in the GATE framework [2].

To match user preferences with the object representation, both represented by weighted concepts of our domain ontology, one can distinguish between several approaches in the information retrieval research community: (a) Graph-based approaches including graph traversals (e.g. [1]), adaptations of existing link-based approaches from standard IR (e.g. [7]) as well as spreading activation based approaches (e.g. [16]). (b) RDFS/OWL Reasoning approaches (e.g. [5]). (c) Approaches which consider concept hierarchies for ranking as well as concept-similarity based approaches (e.g. [4]) or (d) Vector-space model based approaches (e.g. [3]). All of these approaches do not have to face the problem of taking the location context into account.

6. CONCLUSIONS

We have described the information extraction for the semantic tourist recommender system Sightsplanner. We presented the main challenges, which include data scraping from various information sources, normalizing data into our ontology, dealing with multi-lingual data, matching objects from different data sources. Along with the description of the mentioned problems, we have provided our current solutions. Our system can handle proprietary formats and missing languages from web pages. The system uses keyword extraction from description in different languages and adds corresponding types to the object. We also have presented the Merger, which can find the similarity between the objects. Based on the similarity value the system decides whether the two objects represent the same physical object.

As we are expanding our system and adding new data sources, we have to deal with the problems we currently have. The more cities we have, the more we have to deal with multilingual content. The missing information in English makes the data quality poor for a foreign tourist who does not understand the local language. We have offered a solution to translate information automatically. Currently, the quality of translation is poor, but in most cases the tourist understands the main points of the description. An automatic search of the English information from the web could yield in better results. We are investigating the technical solution for that idea.

Object merging quality becomes very important, if the same object can be retrieved from a large number of data sources. In our current implementation most of the objects can only come from maximum of two different sources. If the matching is missed, there will be one duplicate object. For larger cities, the number of data sources can be ten times higher than we currently have. If the matching fails, there can be tens of duplicate objects. The opposite result - some different objects are merged by the system - will also be intolerable. Our current Merger is optimised for our current data set. When we try the same configuration in different city with different languages, the result can be worse. As a future plan we will apply our merger to another city and based on the test data we can modify our implementation. Our current solution is optimized for Tallinn and probably would yield poor results when applied to other cities. Therefore we will look into the machine learning techniques to better deal with disambiguation for different cities.

7. ACKNOWLEDGEMENTS

This research has been supported by European Regional Development Fund. This work has been funded by the European Commission as part of as part of the FP7 Marie Curie IAPP project TEAM (grant no. 251514).

8. REFERENCES

[1] C. Bizer, S. Auer, G. Kobilarov, J. Lehmann, and R. Cyganiak. DBpedia Querying Wikipedia like a Database. *World Wide Web Internet And Web Information Systems*, (Banff, Alberta, Canada):1–13, 2007.

[2] K. Bontcheva, D. Maynard, V. Tablan, and H. Cunningham. Gate: A unicode-based infrastructure supporting multilingual information extraction. In *In Proceedings of Workshop on Information Extraction for Slavonic and other Central and Eastern European Languages (IESL'03), Borovets*, 2003.

[3] P. Castells, M. Fernandez, and D. Vallet. An Adaptation of the Vector-Space Model for Ontology-Based Information Retrieval. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):261, 2007.

[4] O. Corby, R. Dieng-kuntz, and C. Faron-zucker. Querying the semantic web with the corese search engine. pages 705–709. IOS Press, 2004.

[5] J. Davies and R. Weeks. QuizRDF: Search Technology for the Semantic Web. *37th Annual Hawaii International Conference on System Sciences 2004 Proceedings of the*, 00(C):40112, 2004.

[6] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. SemTag and Seeker: Bootstrapping the semantic web via automated semantic annotation. *Proceedings of the 12th international conference on World Wide Web*, (Section 2):178–186, 2003.

[7] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, volume 1, pages 652–659. ACM, 2004.

[8] Google maps api web services, 2011. http://code.google.com/intl/et/apis/maps/documentation/geocoding/.

[9] Google maps javascript api v2, 2011. http://code.google.com/intl/et/apis/maps/documentation/javascript/v2/overlays.html#Google_Maps_Coordinates.

[10] Google translate api, 2011. http://code.google.com/intl/et/apis/language/translate/overview.html.

[11] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff. Semantic annotation, indexing, and retrieval. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2(1):49–79, Dec. 2004.

[12] Postgresql, 2011. http://www.postgresql.org/.

[13] Resource description framework (rdf): Concepts and abstract syntax, 2004. http://www.w3.org/TR/rdf-concepts/.

[14] Regio technical specification, 2011. http://www.regio.ee.

[15] M. Resende and C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 219–249. Springer New York, 2003.

[16] C. Rocha, D. Schwabe, and M. P. Aragao. A hybrid approach for searching in the semantic web. *Proceedings of the 13th conference on World Wide Web - WWW '04*, page 374, 2004.

[17] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. V. Oudheusden. The city trip planner: An expert system for tourists. *Expert Systems with Applications*, 38(6):6540–6546, 2011.

[18] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011.

[19] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. MnM: Ontology Driven Semi-Automatic and Automatic Support for Semantic Markup. *Knowledge Engineering and Knowledge Management Ontologies and the Semantic Web*, pages 379–391, 2002.

[20] Xml path language, 1999. http://www.w3.org/TR/xpath/.

Appendix 6

VI

A. Luberg, T. Tammet, and P. Jarv. Extended triple store structure used in recommender system. In *2011 22nd International Workshop on Database and Expert Systems Applications*, pages 539–543, Aug 2011

Extended triple store structure used in recommender system

Ago Luberg
Eliko Competence Center
Tallinn, Estonia
Teaduspargi 6/2
Tallinn University of Technology
Tallinn, Estonia
Ehitajate tee 5
Email: ago.luberg@eliko.ee

Tanel Tammet
Eliko Competence Center
Tallinn, Estonia
Teaduspargi 6/2
Tallinn University of Technology
Tallinn, Estonia
Ehitajate tee 5
Email: tammet@staff.ttu.ee

Priit Järv
Eliko Competence Center
Tallinn, Estonia
Teaduspargi 6/2
Tallinn University of Technology
Tallinn, Estonia
Ehitajate tee 5
Email: priit@cc.ttu.ee

***Abstract*—We present a database structure and main algorithms for a tourism recommender system. The presented structures are implemented in the Sightsplanner system, able to create plans for visiting interesting objects and events on the trip, based on the personal preference profile. The database structure can be seen as RDF extended with metainformation fields. Every fact in the database has a confidence score, interpreted as the probability. The score is used to deduct new facts and recommend objects to tourists.**

***Index Terms*—Recommender system; database; reasoning; RDF**

I. Introduction

The papers about recommender systems and algorithms typically avoid the issues of data storage and encoding. We attempt to fill this gap by presenting the schemaless database structure along with the custom in-memory database, integrated with the ontology-oriented reasoner. Everything described in the paper is a part of an already functioning tourism recommender system for Tallinn: www.sightsplanner.com.

The paper is structured as follows. First a brief overview of the recommender system itself is given. In sections III and IV we will describe our schemaless structure both in PostgreSQL and in our main memory database. We have also prepared a short performance test in section V and the related work overview in section VI.

II. Tourism Recommender System

Sightsplanner is a semantic recommender and route composer system for tourists. A tourist can specify her location, time of the visit and her preferences about different types of objects and events. Based on the created profile, the recommender finds interesting objects for the given user. For each found object a ranking score is found. The objects that the user probably likes have a higher score and vice versa. The objects and events are organized into a timetable based on their location and time. Finally the trip route will be presented to the user. The tourist has an option to modify the suggested objects and to create a more suitable timetable.

The recommendation process starts with loading the tourism objects from the in-memory database, applying an object filter: objects that don't have required properties (coordinates, opening/closing times), belong to a different locale or are not accessible during the visit duration are ignored. For each loaded object the recommender assigns a ranking score that is calculated by combining the *#type* properties of the object with the preferences from the user profile. The score calculation algorithm is presented in Algorithm 1.

Algorithm 1 Object score calculation

```
1: Objects ← all_objects
2: Profile ← all_user_interests
3: for all o ∈ Objects do
4:     Matches ← {}
5:     for all r ∈ Profile do
6:         for all p ∈ o_properties do
7:             if p_name = r_name then
8:                 Matches ← Matches ∪ {r_score * p_score}
9:                 break
10:            end if
11:        end for
12:    end for
13:    Matches ← sort_descending(|m|, m ∈ Matches)
       ▷ sort matched elements by absolute value, preserv the
       actual value in result
14:    sum ← 0
15:    for i ← 1, size(Matches) do
16:        sum ← sum + Matches_i/2^(i−1)
17:    end for
18:    o_score ← sum
19: end for
```

The array of objects is sorted in descending order of the calculated ranking scores. The recommender then runs the planning algorithm that arranges the objects in a timetable for the user while attempting to maximise the individual object rankings that are being included in the recommendation. There are several constraints to the timetable that need to be satisfied - the tourism objects have opening times (therefore, the optimization problem as modelled in Sightsplanner may

be classified as an instance of OPTW [1] problem) and each of the preferred categories that was present in the user profile should be represented (due to the latter requirement the OPTW problem in Sightsplanner could be classified as multi-objective optimization, however we have chosen to aggregate this criteria into the overall score of the recommended timetable).

The planning problem is solved using a meta-heuristic search based on GRASP [2]. The method of search was originally described by Vansteenwegen et al, who employed this type of heuristic in their Citytripplanner recommender [3] . Sightsplanner uses a similar approach, where the candidate objects chosen in GRASP iterations are added incrementally into the current solution, therefore having no distinct local search phase. The balance criteria that requires objects from varied categories to be present in candidate solutions is used to additionally guide the selection of items in the GRASP search.

When computing the solution to the maximisation problem, the *#latitude* and *#longitude* properties of the objects are used to estimate the time required to move from one candidate object to another. The *#opening_time* properties are used to determine, whether an object is accessible at any given moment.

The meta-heuristic search produces a number of candidate solutions which are ranked using an aggregate score which is based on the average score of the objects included and the balance criteria. The top ranking solution is returned to the user. The output of the recommender is a timetable of objects, containing a Unified Resource Identifier (URI), arrival and departure time and method of travel from the previous location for each object.

The PostgreSQL database of Sightsplanner contains all the known facts about the tourism objects. Periodically, the data needed for computing the recommendation is propagated to the in-memory database. The PostgreSQL database itself contains additional properties that are used for displaying the recommendation to the user. This data is retrieved using the object URIs from the solution.

III. PostgreSQL

Before we can give a more detailed look into database, we have to describe how the data is obtained. For the Tallinn recommender system we are currently scraping six different web portals with completely different structures. Raw data obtained from the portals is then normalized for our system. Some of the objects in the recommender are one-time events (concerts, performances etc) which are scraped from the web daily. Another categeory of objects is formed by places of interest (POIs) with mostly static information - nothing changes in months or even years. The POI objects are updated typically once a month.

We have implemented a specific scraper algorithm for every site based on the structure of the web page. The system also looks for existing objects in the database before adding new data. Objects are compared by different properties (location, title, categories, opening times and other meta data) and the similarity is found. The comparison algorithm has different weights for different comparators. For example, the location comparison can have higher impact on the similarity outcome than the title comparison (different fast food eating places with the same name). The real weight values depend on the objects which are compared - events have totally different comparison conditions than static POIs. If the found similarity exceeds a certain threshold, the objects are merged - the new scraped data is added to the existing matching object. The merging is very important for multilingual data. Currently we are gathering data in English and in Estonian. We scrape some pages, which only present information in one of those languages. Without merging, we would end up having two instances of the same object - one in Estonian, one in English. With the help of location, categories, phone number, title etc. we can find the duplicate objects and merge those.

In addition to the scraping algorithms, we are using several manual data manipulation methods. The quality of scraped data cannot be trusted 100% and some important events, POIs and their information are not available for scraping. Hence the need to manually enhance scraped data. For example, the popularity information for most of the objects is not available from the web. Therefore all the objects are associated with the popularity value, which indicates the estimation of daily tourist visits for the given object. We have also added a significant amount of POI object types (like church, museum, subtypes of these etc) manually.

As said, we use both the traditional database (PostgreSQL) and the in-memory database. All the data manipulation for initial insertion, enhancing and modification takes place in PostgreSQL. The automated type derivation algorithms as well as the ranking and scheduling algorithms called for user requests operate on the in-memory database. Therefore we have set up an automatic synchronization of two databases. The information from PostgreSQL is imported into the memory database. This takes place every night, since we scrape new information about events nightly.

The database structure is based on the RDF model, extending the traditional triplet fields (*object*, *property*, *value*) with metainformation about the triplet. The reason for the extension is to avoid reification. The system has one main data table for the records. All the values and URIs are stored in the second, *string* table. The description of the main data table can be seen in the table I.

The *score* field shows the confidence or probability of the fact within range [0; 1]. For example, a fact can say that "church X has a name Oleviste" with the score 0.9. Another fact about the same object X may say "object X has a name Niguliste" with a lower score 0.5, intuitively meaning that the church X probably has a name "Oleviste".

The system uses scores to calculate the recommended objects for each user request. As a typical example of score usage, consider object types. Each object may have several different types, each associated with the probability score. For example: "object X has a type *#church*" with score 1.0, "object X has a type *#architecture*" with score 0.7, "object X has a type *#museum*" with score 0.2. If the tourist likes

TABLE I
THE MAIN TABLE STRUCTURE

Data type in PostgreSQL	Data type in memory DB	Description
integer	int	Unique row id
integer	uri	Object URI (reference to string table id)
integer	uri	Property URI (reference to string table id)
integer	depending on the data type	Value URI (reference to string table id)
real	double	Score/strength of given connection/association
integer	uri	Source URI of given fact (reference to string table id)
timestamp	int	Timestamp when this fact was created
datetime	int	Fact is valid starting from this datetime
datetime	int	Fact is valid until this datetime
integer	int	Whether this row is private (not visible) info

churches and doesn't like museums so much, object X would be recommended for her. But if the tourist likes museums and doesn't like churches so much, then the object is unlikely to be recommended for her.

The *source* field indicates the name of the source for the given fact. Facts come from different sources. Some are scraped from the web, some are manually inserted, others are reasoned from other facts etc. Different methods result in a different source name. For example, we often get different names for the same object from different web portals. Using source fields, we can make difference between the facts and favour one over the other.

The *timestamp* field shows the date and the time of the creation of the fact.

The *validity start* and *validity end* fields show the period during which the fact is valid. If the fields are omitted, then the fact is valid all the time.

Privacy indicates the level of invisibility. Default is NULL, which means the fact is visible or public. Higher value would mean that the fact is hidden (not visible) and therefore cannot be used for all purposes.

In PostgreSQL, the main table does not hold direct values for *object*, *property*, *value* and *source*. All those fields are references to the *string* table, which holds the real values. As the value field in string table holds text, we also need the additional *data type* field. We use several different data types (*string*, *URI*, *date*, *time*, *integer* etc). For example, *String* is just a text value like the name "Oleviste". *URI* is represented by a text value, typically started by the "#" symbol. The *object_id*, *property_id* and *source_id* fields reference to *URIs*. The *value_id* can also reference to *URI* on certain cases. For example, the type of an object described above in the examples is an *URI*. Each fact with a string value references to a different string table record. However, all the same *URIs* point to the same string table record. Other data types act as strings - there can be duplicate values.

Our in-memory database uses a similar structure for storing facts. Most of the values from PostgreSQL are stored in one main table. Only long strings and URIs are stored separately while a pointer is stored in the main table.

Our in-memory database allows different rows to have a different number of fields with different types. However, in our recommender application we do not use this functionality: all the in-memory database rows have the same number of fields with the same types.

IV. THE REASONER WITH THE SHARED MEMORY DATABASE

The reasoner uses rules to deduct new facts about the objects. We are using RIF [4] style rules with an additional confidence score attached to each rule. Similarly, confidence scores are also attached to each fact in the database.

The main goal of the rules is to derive confidence-annotated new additional types for data objects from both given types and additional information, like the name of the object.

Consider these three rules:
0.9: fact(?X type architecture) :- fact(?X type church).
0.9: fact(?X type drinking_place) :- fact(?X type bar).
0.7: fact(?X type eating_place) :- fact(?X type bar).

The first rule indicates that if an object has a type *church* with the *confidence score* N, it is also an *architecture* with the confidence score $0.9 * N$. Similarly, a bar will be - with a high confidence - a drinking place, but also - with a lower confidence - an eating place.

The reasoning engine employed by the system has to handle a large amount of data - all properties of all POIs - plus a relatively large ontology. The same holds for the suggestion engine, which has to quickly access all the original and all the derived data about POIs and user preferences.

Object properties do not depend on the user profile. Our reasoner generates all the properties with confidence scores for each object based on the rules. As the rules and the data do not change often, the property generation process is run once every night. This saves time for the suggestion requests, which use pre-calculated facts.

We employ a principle that all - or relevant for this day - data from the relational database is loaded into the fast in-memory database handled by the prover. Changes in the relational database are synchronised into the in-memory database using

TABLE II
REAL TIME USAGE IN MS

Function	Query	Pure query
avg	137.28	110.51
stdev	0.39	0.44

TABLE III
MATCHED OBJECTS

Function	Loaded	Accessible	Matched
min	1551	474	44
max	1551	532	528

scripts. All the derivations conducted by the reasoner use only the in-memory database, which serves as a native data store of the reasoner.

As explained before, we have built a new reasoner-equipped fast in-memory database to serve our needs. This in-memory database is implemented in shared memory as a library, without any continuously running processes. Each update or query to the database is performed as a call to the corresponding function. In order to enable parallel queries, we have implemented a fast global read/write lock using spinlocks. Those prohibit simultaneous data addition and reading from the global data area. Each query process has an additional local non-shared data area for query input and temporary results. Thus it is possible to conduct several derivation queries in parallel, avoiding the need to copy common data to all the separate simultaneous derivation processes.

As the shared memory must be able to accept all kinds of data and all kinds of schemas we may use in our relational database, it is "schemaless" in the sense of not having the classical schema of relational databases.

The shared memory database must be also suited for keeping RDF data in the efficient manner: therefore we have implemented native support for RDF datatypes like URIs, XML literals, strings with the language attribute etc. However, the database - and the reasoner - is not limited to RDF. It is capable of holding arbitrarily long tuples of arbitrary data. Hence our data kept in the relational database can be imported into the shared memory database in a straightforward manner, without the need to reify, i.e. each database record is converted to a single shared memory database record with exactly the same number of fields.

Our reasoner is built on the first order resolution-based theorem prover Gandalf [5], employing and extending selected algorithms from this prover. Additionally we are using a highly specialised C code module, which is essentially a hand-compiled set of core rules performing the same task as the reasoner. The hand-coded module is highly efficient, but lacks the flexibility of the rule system.

We employ both conventional in-memory database indexing algorithms (T-trees and hashes on fields and field vectors) combined with the indexing algorithms conventionally used in provers, like the widely used discrimination tree index for unit subsumption and unit deletion.

V. PERFORMANCE

To give an overview of the performance of the memory database, we measured the average query time needed to retrieve all the data that is used to make one recommendation. 15 different user profiles where used. These profiles represent typical usage, including interest in a single category of objects as well as a wide range of categories. User profile presets from Sightsplanner web interface were included. Each test was repeated 5 times. The database included 1551 attractions which represents a realistic size for a localized tourism recommender database.

We have measured two different times: *query* shows the total time spent on retrieving the objects by the recommender. Because this also includes the object score calculation, the second measure *pure query* shows only the database querying time. The tests were run on a Core2Duo T8100 system, with 100MB of shared memory used by the database. Times are shown in Table II. The first row shows the average running time of tasks in milliseconds, whereas the second row shows standard deviation. As the standard deviation is low, we can say that query time did not depend significantly on the user profile.

The results of the queries are given in Table III: column *loaded* shows the total object count in database, *accessible* presents the number of objects, which were found to be opened during the user visit and *matched* shows the number which gave a positive match to the user profile. The number of objects loaded from the memory database stayed constant throughout the test. About one third of all the objects were opened during user's visit time, the small variations were due to the different start times and durations in different profiles. The number of objects matched to the profile exhibited the most variation.

VI. RELATED WORK

There are several triple store databases which use the relational database management systems (RDBMS) for data storage. We will review some of them as the schema is available and can be easily compared with our system. There are other triple stores for which the underlying data storage is complicated or unknown. Therefore it is hard to draw parallels with our system.

First, we look Jena [6] - Semantic Web programmers toolkit, open-source, implemented in Java. Jena version 1 used very similar approach to ours. They have a table for statements which has 4 fields: subject reference, predicate reference, object URI reference, object literal reference. Subject, predicate and object URI refer to resources table, object literal refers to literals table. Instead of resources and literals tables we have one string table in PostgreSQL which combines values and URIs. In Jena2, subject, predicate and object fields in statement table store the value instead of referring to literals or resources table. They have denormalized the structure in order to save time querying data, but they have increased the database size. Jena1 did not duplicate URIs, Jena2 schema

does duplicate subject URIs. To lower the space usage, longer URIs and values are still stored in resources and literals tables.

Another triple store engine we look into is Sesame [7] - framework for processing RDF data. They also have an option to store data in RDBMS along with memory and native configuration. The schema has four main tables: Class, Property, SubClass and SubProperty. For every new class or property a new table is created which stores all its instances. They also have resources table, which stores values of every resource and literal to minimize the size of the database. Adding tables to the database does not follow schemaless structure that we try to have in our system, therefore this path is not an option.

We also include Virtuoso [8] in our review. Virtuoso is universal platform for data integration and management. It has many functionalities, data storage is one of them. Virtuoso DBMS also has RDF support, which basically has one data table with four fields: graph, predicate, subject and object.

Here we have drawn some parallels to our database structure and few popular public database engines. As we have compared triple data stores, then the basic idea is the same. The main difference is how to handle different RDF objects (value in our structure) and how to store often used properties. In our system we store all the literals and resources in the string table, which also has fields to indicate the value language and type: an URI, a string, an integer etc. We have extended our data table to add fields like score, source, timestamp and others. Score is the most important of them as it is used both by the reasoner and the suggestion engine. In the memory database, where the actual reasoning and querying takes place, we have only one table. All the data from the PostgreSQL string table is added to this one table.

VII. Conclusion

We have given a brief overview of the database structure and usage in our recommender system. The presented schemaless structure in PostgreSQL and in our memory database have proven to be efficient and useful. With every fact in the database we store the score or probability which indicates the strength of the fact. The score is used to deduct new types for POIs and to calculate the object preference score for each user profile. The probabilistic score gives an opportunity to conduct probabilistic reasoning tailored to the tourism application.

Some future directions for our database and reasoner part include SPARQL [9] implementation and probabilistic reasoning improvements. Currently we have very limited needs for querying data from the database, hence there is no strong need for using SPARQL. However, the SPARQL implementation would give us an opportunity to compare our database directly with the other implementations using benchmark frameworks like The Berlin SPARQL Benchmark [10].

Acknowledgment

This research has been supported by European Regional Development Fund.

References

[1] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, "The orienteering problem: A survey," *European Journal of Operational Research*, vol. 209, no. 1, pp. 1–10, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221710002973

[2] M. Resende and C. Ribeiro, "Greedy randomized adaptive search procedures," in *Handbook of Metaheuristics*, ser. International Series in Operations Research & Management Science, F. Glover and G. Kochenberger, Eds. Springer New York, 2003, vol. 57, pp. 219–249.

[3] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. V. Oudheusden, "The city trip planner: An expert system for tourists," *Expert Systems with Applications*, vol. 38, no. 6, pp. 6540–6546, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417410013230

[4] (2011) Rule interchange format. [Online]. Available: http://www.w3.org/2005/rules/wiki/RIF_Working_Group

[5] T. Tammet, "Gandalf," *Journal of Automated Reasoning*, vol. 18, pp. 199–204, 1997, 10.1023/A:1005887414560. [Online]. Available: http://dx.doi.org/10.1023/A:1005887414560

[6] K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds, "Efficient rdf storage and retrieval in jena2," in *EXPLOITING HYPERLINKS 349*, 2003, pp. 35–43.

[7] J. Broekstra, A. Kampman, and F. van Harmelen, "Sesame: A generic architecture for storing and querying RDF and RDF Schema," in *Proceedings of the first Int'l Semantic Web Conference (ISWC 2002)*, ser. Lecture Notes in Computer Science, I. Horrocks and J. Hendler, Eds., vol. 2342. Springer Verlag, May 2002, pp. 54–68. [Online]. Available: http://www.springerlink.com/content/hj139wcxnl9aer5u

[8] O. Erling and I. Mikhailov, "Rdf support in the virtuoso dbms." in *Conference on Social Semantic Web*, ser. LNI, S. Auer, C. Bizer, C. Müller, and A. V. Zhdanova, Eds., vol. 113. GI, 2007, pp. 59–68. [Online]. Available: http://dblp.uni-trier.de/db/conf/cssw/cssw2007.html

[9] (2011) Sparql query language for rdf. [Online]. Available: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

[10] C. Bizer and A. Schultz, "The berlin sparql benchmark," *International Journal On Semantic Web and Information Systems*, 2009.

Appendix 7

VII

A. Luberg, T. Tammet, and P. Järv. Smart city: A rule-based tourist recommendation system. In R. Law, M. Fuchs, and F. Ricci, editors, *Information and Communication Technologies in Tourism 2011*, pages 51–62, Vienna, 2011. Springer Vienna

Smart City: a Rule-based Tourist Recommendation System

Ago Luberg[a],
Tanel Tammet[a], and
Priit Järv[a]

Eliko Competence Center,
Tallinn, Estonia
info@eliko.ee

Abstract

We are presenting the architecture of a personalised recommendation system using probabilistic reasoning. The system creates plans for visiting interesting objects and events on the trip, based on the personal preference profile. We will provide a short description of the system architecture, its main components and the probabilistic reasoner along with a small ruleset example with score calculations.

The described architecture and components comprise the first iteration of the Smart City system.

Keywords: Recommendation engine; tourism; semantic web; rule-based reasoning.

1 Introduction

People planning a trip often start investigations by looking for materials on the Internet. The user can find a large amount of information about the city and its popular tourism objects, but it takes a long time to select the most interesting objects and create a good plan for a visit. For example, consider a tourist planning a two-day visit to Paris. If she wants to put together an organized plan for those days, she will first have to find all the potentially interesting objects in Paris. Then she will have to decide which ones she would like to visit. After that she will have to organize the selected objects into a schedule. It may turn out that some of the objects do not fit into the time schedule, in which case she will have to select different objects. All this takes a lot of effort.

Our project focuses on automating the object selection and visit schedule creation. Smart City (SC) is a semantic recommender and route composer system for tourists. A tourist can specify her location, time of the visit and her preferences about different types of objects and events. Based on the created profile, the suggestion engine finds interesting objects for the given user. For each found object a score of "interestingness" is found. The objects that the user probably likes have a higher score and vice versa. The objects and events are organized into a timetable based on their location and time. Finally the schedule and the trip route will be presented to the user. The tourist has an option to modify the suggested objects and to create a more suitable timetable.

The system is currently undergoing the public test phase (see http://gn61.zone.eu/gui/) and will be properly deployed at the beginning of 2011 with the initial launch in Tallinn, Estonia, to be followed by several cities outside Estonia. The adequate business analysis of the first deployment phase can be given after the system has been fully exploited for at least several months/half a year. However, we would like to point out our initial business assumptions and foreseeable business implications.

First of all, the system is not targeting the functionality of traditional tourism sites, hence it would obviously make sense to deploy the system as a search/planning component of a larger tourism site. This is exactly the plan for the initial deployment in Tallinn, where the system should be a new component of the existing site http://www.tourism.tallinn.ee/ . This gives us an obvious way to direct visitors to our site and an additional benefit of having better organisational means for both (a) competing for advertiser funding and (b) accessing existing databases, hence somewhat diminishing the need to do complex html scraping for data.

The direct benefit to consumers is obvious: easier planning of the trip and better search possibilities, hopefully leading to a better practical experience and satisfaction on location. However, we believe that the indirect benefits are even more important. First of all, the tourist will have far better means for finding specific objects of interest, even if they are not so "mainstream" or hugely popular. Concerts and performances would be a perfect example: most tourist sites do not directly integrate the concert/performance listings and if they do, it is hard to find the personally interesting genre and even harder to fit the performance into the trip plan, for example, by selecting a restaurant based on the combination of a performer, performance time, restaurant type and suitable location and match in the trip schedule.

The wider adoption of the system would likely benefit the wider spectrum of service providers in the city: there would be more tourists at less "mainstream" locations and consequentially somewhat fewer tourists at the main tourism sites. We believe this would also give a better experience to the tourist, who would get a better "blend-in" feeling, diminishing the often negative experience of seeing only highly "touristic" places.

The Smart City project uses experience from the Smartmuseum project (Liiv, Tammet, Ruotsalo, & Kuusik, 2009) and applies some of the ideas deployed in the Smartmuseum project to the outside tourist scenario.

2 Related Work

In the recent decade a significant amount of research has been conducted in the field of recommendation systems. There are several overviews of different techniques and methodologies (Adomavicius & Tuzhilin, 2005). The most widely known technique used in recommendation systems is Collaborative Filtering (CF) (Pitsilis & Marshall, 2008; Goldberg, Nichols, Oki, & Terry, 1992). For example, Amazon.com and Jester (joke recommendation system) both use CF (Adomavicius & Tuzhilin, 2005).

Collaborative filtering systems can produce personal recommendations by computing the similarity between the user preference and the one of the other user. Typically, a subgroup of users of the similar interests to the user is selected. The given subgroup preferences are then used to recommend options which do not have a personal opinion.

As there are several problems concerning CF ("cold start" problem - a new user or a new item is added without connections etc.) (Adomavicius & Tuzhilin, 2005), hybrid or other techniques are used (Sebastia, Garcia, Onaindia, & Guzman, 2008). A content-based (contextual) recommendation system is described in (Gonzalo-Alonso, Juan, Garcí-A-Hortelano, & Iglesias, 2009). Contextual recommendation is based on finding similarities in the user profile and candidate items. Obviously, the quality of the recommendations depends on the amount of relevant information about items and the profiles.

Recommendation system authors often point out the necessity to use ontologies or relatively simple taxonomies, see (Sebastia, Garcia, Onaindia, & Guzman, 2008). A common ontology gives an opportunity to group the necessary terms and define the relationships between the related terms. A brief overview on ontologies and the Semantic Web in E-Tourism is presented in (Siricharoen, 2008). Chiu, Yueh, fung Leung, & Hung (2009) present a specific application-oriented ontology.

Several systems use probabilities or a fuzzy criteria to recommend the items. In article by young Kang, Kim, & Cho (2006) a similarity coefficient is offered to find the best suitable point of interests for the given user. The similarity is calculated by the profile and the item vectors. The similarity is larger if the angle between the vectors is smaller. Ciaramella, Cimino, Lazzerini, & Marcelloni (2009) have presented an uncertainty of situations based on the contextual conditions.

Golovin, & Rahm (2005) are creating rules by the recommendation process. Given rules are stored in the recommendation database. Another paper (Fuhr, 1999) describes probabilistic Datalog. The system is able to cope with independent events. Calculating the probabilities of the independent events is far less complex than calculating probabilities of dependent events. In case the degree of dependency is not known, intervals are required to present the probabilistic range of the event. Fuhr (1999) also describes an implementation of a working system based on the presented algorithms and principles of probabilistic Datalog.

As we have already mentioned, the work originating from the Smartmuseum (SM) (Liiv, Tammet, Ruotsalo, & Kuusik, 2009) project is used in the SC project. SM is a recommendation system for museums. The tourist creates a profile about her interests. Similarly to the SC project, different objects inside a museum are recommended. However, the architecture and the recommendation methods are different from the methods employed in the current SC project.

3 Overview of the Recommendation System

The browser-based user interface has only two main views. The figure 1 shows the start page of the system. The user defines her visit time and degree of interest in five different main topics. Four predefined interest combinations are available. In the figure the "Walk in the city" is selected. After submitting the form a suggested trip plan is calculated and shown (figure 2). The itinerary contains all the suggested objects and walking time between them. Each object in the list can be removed or the visit duration can be changed. All the objects are also presented on the map below the listing.

The tourist has an option to get more objects into the plan: the system will try to fit new objects into the existing timetable.

A more detailed description of recommendation process and schedule creation is provided in a separate section *Score calculation.*

The Smart City system itself is divided into several different components. We will present an overview of the components and their role in the system. A more technical and detailed description of critical modules can be found in the next section.

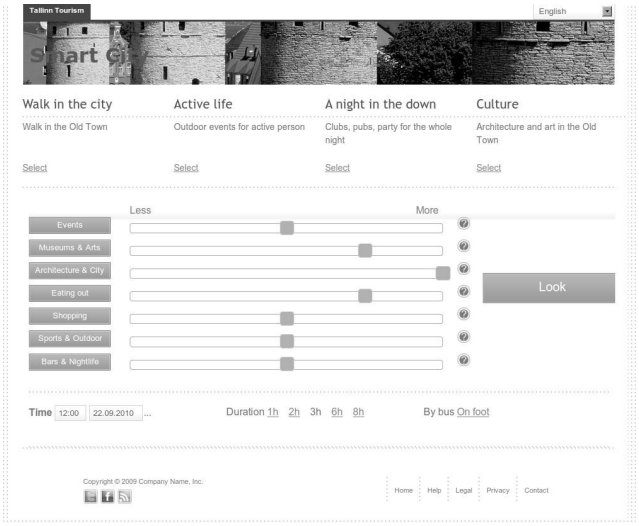

Fig. 1. The start page of the recommender system

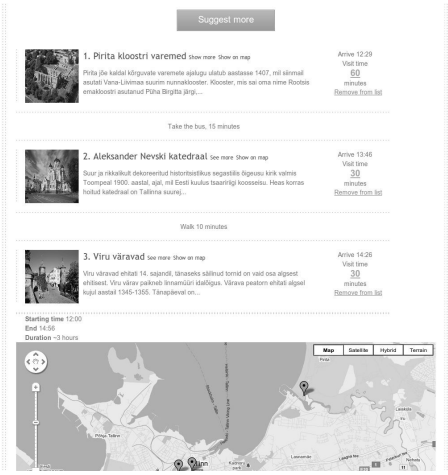

Fig. 2. Suggested schedule view

The central component in our architecture is the Control Centre (CC). The CC is responsible for communicating with the different components and generating the User Interface (UI). The CC is connected directly to the persistent database which holds all the information about the tourism objects. The data is gathered automatically from the different content providers. Some publish web services, some provide access to their database or data files. For each source provider we use importer script designed specifically for the needs. As the sources are different, the problem of data interoperability has to be solved. We have adopted our own structure for the storage and convert the original data formats to this structure. We also have created our own ontology for the terms and objects.

The CC is connected to the suggestion engine (SE), which is responsible for creating a trip plan. The CC sends all the information about the user preferences to the SE. JSON (JavaScript Object Notation) is used to transport information between those two components. The SE returns a list of the suggested objects along with times. The CC presents the list to the user. If changes are made into the plan (user has removed some objects or altered the visit time for the objects) the CC sends the modified plan to the SE. Thus SE knows about the changes and can leave the user modifications unaffected.

The suggestion engine uses a rule-based reasoner with an integrated shared memory database to find all the scores for the objects. The reasoner employs the user preferences and rules to add scores to the objects. The objects with scores for the given user are returned to the SE. The reasoner uses the memory database as its knowledge base. The memory database should always have the latest information

about the objects, therefore all the necessary data is synchronized from the persistent database. A more detailed preview of the reasoner and the score calculation is presented in the next sections.

4 Technical Overview of Components

The Smart City recommendation system uses an ontology to classify both the places of interest and event (POI) types and user interests. Every term used in the rules is presented in the ontology. We have used different online vocabularies to assist our work: (Wordnet Princeton University, 2010) and Getty (The J. Paul Getty Trust, 2010).

Table 1. The main table structure in PostgreSQL

Field	**Description**
id	Unique row id.
object	Object URI of N3.
property	Property URI of N3.
value	Value URI of N3.
connection_score	Score/strength of the fact in range [0..1].
source	Source of the given fact.
timestamp	Date and time when the fact was created.
validity_start	The fact is valid starting from this date and time.
validity_end	The fact is valid until this date and time.
privacy	Whether the fact is private (not visible).

Data providers use different formats to present their information. A separate import module is created for each source. For example, a custom structured XML is converted into our ontology. We have imported data from the Visit Estonia system, presented to us as XML files. We are also using web pages as data sources (Tallinn Tourism web page and local events database mistoimub.ee). A web scraper configuration is created for different sources. Since some sources do not use the structured data presentation, we are planning to use content-based classification in the future. For example, a concert might not have an annotation about the genre. However, the description of the given event can be parsed to find certain words describing genres. The performing bands can also be used. If the bands are annotated in the system and each has a set of genres provided, this information will be used to classify the event.

All the information is stored in the persistent database. Smart City uses PostgreSQL with an essentially schemaless structure, extending a triple storage structure RDF used often in the Semantic Web context. Each fact is a tuple containing an object, property, value and additional information. The main table structure is shown in the table 1.

5 The Reasoner With the Shared Memory Database

The reasoning engine employed by the system has to handle a large amount of data - all properties of all POI-s - plus a relatively large ontology. The same holds for the suggestion engine, which has to quickly access all the original and all the derived data about POI-s and user preferences.

Fetching all this data from the conventional relational database takes too much time. In particular, it is hopeless to run the reasoning engine by querying all the potential premises of rules from the conventional database for each rule application during derivation. The reasoning engines are typically built on specialised efficient main-memory data stores.

Hence we employ a principle that all - or relevant for this day - data from the relational database is loaded into the fast in-memory database handled by the prover. Changes in the relational database are synchronised into the in-memory database using scripts. All the derivations conducted by the reasoner use only the in-memory database, which serves as a native data store of the reasoner.

Additionally, the reasoner should be able to serve a number of queries simultaneously, each one conducting a complex derivation. Hence the common set of facts and rules should be kept in the same memory store for all the queries, capable of running in parallel.

We have built a new reasoner-equipped fast in-memory database to serve our needs. The in-memory database is implemented in shared memory as a library, without any continuously running processes. Each update or query to the database is performed as a call to the corresponding function. In order to enable parallel queries, we have implemented a fast global read/write lock using spinlocks. Those prohibit simultaneous data addition and reading from the global data area. Each query process has an additional local non-shared data area for query input and temporary results. Thus it is possible to conduct several derivation queries in parallel, avoiding the need to copy common data to all the separate simultaneous derivation processes.

As the shared memory must be able to accept all kinds of data and all kinds of schemas we may use in our relational database, it is "schemaless" in the sense of not having the classical schema of relational databases.

The shared memory database must be also suited for keeping RDF data in the efficient manner: therefore we have implemented native support for RDF datatypes

like URI-s, XML literals, strings with the language attribute etc. However, the database - and the reasoner - is not limited to RDF. It is capable of holding arbitrarily long tuples of arbitrary data. Hence our data kept in the relational database can be imported into the shared memory database in a straightforward manner, without the need to reify, i.e. each database record is converted to a single shared memory database record with exactly the same number of fields.

Our reasoner is built on the first order resolution-based theorem prover Gandalf (Tammet, 1997), employing and extending selected algorithms from this prover.

The system employs the reasoner in two different ways. First, it regularly completes the database by deriving all the new facts from the rules and input facts. We use a specialised version of a forward-chaining hyper-resolution for this goal. Second, it may perform specific queries, where we employ a version of a binary backward-chaining set-of-support strategy. See (Robinson, & Voronkov, 2001) for the common algorithms employed in first-order automated reasoners.

We employ both conventional in-memory database indexing algorithms (T-trees and hashes on fields and field vectors) combined with the indexing algorithms conventionally used in provers, like the widely used discrimination tree index for unit subsumption and unit deletion. The temporary, local area uses the discrimination tree index while the shared memory database area uses T-trees and hashes.

6 Score Calculation

The Control Centre sends the user profile properties with a *property preference score* to the suggestion engine. The latter will send this information to the reasoner. On the first page of the system's graphical user interface the user can use sliders to indicate her interests in different topics. Each slider will provide a certain *property preference score* for the user profile. We will provide a short example:

user1 profile:
music 70
architecture 40
museums 40
sports 40
food 50

In this example we have five preferences along with the scores. The score indicates the user interest in the given preference within the range [0, 100]. Let us consider two following objects:

object1:
music 50
architecture 30

museums 20
food 10

object2:
music 90
sports 60

Objects have properties with *property confidence scores*. The score indicates the confidence that the given object has this property.

The reasoner uses rules to deduct new facts about the objects. We are using RIF (Rule Interchange Format, 2010) style rules with an additional confidence score. The given score is presented as a number in the range [0,1] and is included to each rule. We will provide a simple example:

fact(?X type architecture 0.9?N) :- fact(?X type church ?N)*

The above rule indicates that if an object has a type *church* with *confidence score N*, it is also an *architecture* with the confidence score *0.9 * N*.

Each object in the database is compared to the given user profile. Based on the *profile property preference score* and the *object property confidence score* the *object preference score* is found.

Object properties do not depend on the user profile. Our reasoner generates all the properties with confidence scores for each object based on the rules. As the rules and the data do not change often, the property generation process is run once every night. This saves time for the suggestion requests, which use pre-calculated facts.

Once the user runs a suggestion query, the profile is passed to the suggestion engine. The reasoner generates the profile with the properties and their *preference scores* based on the rules. After the profile generation, the reasoner calculates the scores for each property available both in the profile and the object. We use a simple product of the two scores as the result. Our two objects will have the following property confidence scores:

object1 for user1
*music 0.7 * 0.5 = 0.35*
*architecture 0.4 * 0.3 = 0.12*
*museums 0.4 * 0.2 = 0.08*
*food 0.5 * 0.1 = 0.05*

object2 for user1
*music 0.7 * 0.9 = 0.63*
*sports 0.4 * 0.6 = 0.24*

All the objects with the *property confidence scores* for the given user are found and SE will calculate the *object preference score*. Our current logic for the calculation is to sum all the *property confidence scores*. So, the *object preference score* for *object1* would be 0.35 + 0.12 + 0.08 + 0.05 = 0.6. *object2* will have a score 0.87.

The suggestion engine computes the *preference scores* for each object. All the objects are passed to the schedule creating process, which consists of the following steps: 1. sort the array of objects by *preference score*, 2. run multiple schedule calculation algorithms on the resulting array of objects, and 3. collect the results and find the result which has the highest overall score.

To maintain a reasonable response time for the suggestion engine service, the schedule calculation step has a limited running time (100ms in SmartCity, currently). Schedule calculation algorithms may use one of two techniques: either process the entire array of objects once or process the array iteratively, increasing the number of objects examined on each iteration. The iterative technique is based on the rationale that the algorithm is guaranteed to return some result, even suboptimal, in early iterations with a low number of objects and then progressively improve the calculated schedule by examining more objects, until the timer expires. Since the array of objects is pre-sorted, early iterations examine the higher-scoring objects.

Three scheduling algorithms have been implemented. *Dummy* is an extremely fast and inaccurate algorithm that returns the top-scoring objects from the array, only ensuring that they are actually open during the time of the visit. *Bruteforce* is an accurate but slow algorithm which attempts to examine all possible combinations of objects, therefore obtaining optimal results on small data sets. *A** is an adaptation of the widely-known path finding algorithm. The cost in conventional *A** is replaced by cost/benefit value, which is calculated as time/score. The "goal" in *A** is to use up the entire time allocated for the schedule - as the algorithm tries to keep the cost as minimum, the score (inverse of cost) will be maximised. This algorithm is between the other two in terms of speed and accuracy. By combining the different algorithms, the scheduler can exploit their individual strengths depending on the situation (number of objects; server load; different profiles etc.).

Preliminary testing on a database of 261 real tourism objects shows that *Bruteforce* algorithm obtains better-scoring schedules in approximately 80% of the tests when the visit time is shorter, while for longer schedules the *A** algorithm obtains higher scores in approximately 50% of the cases and has a higher average score. The *dummy* algorithm is out-performed by the other two in vast majority of the tests. These results are consistent with the expected properties of the individual algorithms.

The results returned by the schedule algorithms are checked for quality. Penalties are assigned to a schedule score based on the interaction of the objects. Following criteria apply: similar objects should not be repeated several times; as many different categories of objects as possible should be represented; eating out and workout-related objects should be limited in number (the tourist has limited capacity for eating and generally does not wish to exert herself excessively). The highest-scoring

schedule, post-quality check, is selected as the recommendation by the scheduler component.

7 Conclusions and Future Work

In this paper we have provided a quick overview of the architecture of the Smart City system. Our project uses ontologies and rules to reason and calculate the interest scores for tourism objects and events. A simple and intuitive user interface makes it easy to find the needed information quickly. We have described the logic of rules and reasoning. We are continuing our research with the goal of improving the suggestion process. New data providers will be connected to import more facts. Along with the new knowledge we will improve our ontology to support the obtained data.

The described system is only a small part of the whole project. We have already planned several additions which would improve the system. One of the planned improvements is saving the user preferences into a persistent profile. This profile will store all the user's history and feedback. The recommendation system can then make suggestions based on user's earlier choices. Also, the engine will not suggest objects which the user has already visited.

While the user is on the trip, she can make changes into the plan and obtain an updated suggestion. For example, the user wants to skip one object or spend more time in one museum. The system will automatically modify the rest of the plan according to the situation. For example, it can find an alternative object to visit instead of the skipped one. The system could also just skip one object so that the user has more time to spend on the previous object or to relax.

Additional data from the social networks can be gathered to complement the profile. The user may have watched videos of classical music in Youtube or has become a fan of a pizza place in Facebook. All this information could be used to improve the user profile. The user does not have to add her interests manually in the cases where this information is available on connected sites.

Our current approach is certainly not final and continuous research on the different topics will proceed. We are actively working on the subject of negative facts (*like* vs *dislike*). Dependent properties could be handled differently from independent properties. In our examples we had an object and a profile with only independent properties, but we already have started working on dependent properties. An ongoing research on how to improve the object score calculation and how to get more balanced schedules will continue.

References

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering, 17*(6), 734-749.

Chiu, D.K.W., Yueh, Y.T.F., fung Leung, H., & Hung, P.C.K. (2009). Towards ubiquitous tourist service coordination and process integration: a collaborative travel agent system architecture with semantic web services. *Information Systems Frontiers, 11*(3), 241-256.

Ciaramella, A., Cimino, M.G., Lazzerini, B., & Marcelloni, F. (2009). Situation-aware mobile service recommendation with fuzzy logic and semantic web. *Intelligent Systems Design and Applications, International Conference on,* 1037-1042.

Fuhr, N. (1999). Probabilistic datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science, 51*(2), 95-100.

Goldberg, D., Nichols, D., Oki, B.M., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM, 35*(12), 61-70.

Golovin, N., & Rahm, E. (2005). Automatic optimization of web recommendations using feedback and ontology graphs. *Web Engineering,* 375-386.

Gonzalo-Alonso, J., Juan, P., Garcí-A-Hortelano, E., & Iglesias, C.A. (2009). A hybrid collaborative filtering system for contextual recommendations in social networks. *DS '09: Proceedings of the 12th International Conference on Discovery Science,* 393-400. Berlin, Heidelberg, Springer-Verlag

young Kang, E., Kim, H., & Cho, J. (2006). Personalization method for tourist point of interest (poi) recommendation. *Knowledge-Based Intelligent Information and Engineering Systems,* 392-400.

Liiv, I., Tammet, T., Ruotsalo, T., & Kuusik, A. (2009). Personalized context-aware recommendations in smartmuseum: Combining semantics with statistics. *Advances in Semantic Processing, International Conference on,* 50-55.

Pitsilis, G., & Marshall, L.F. (2008). Modeling trust for recommender systems using similarity metrics. *Trust Management II Proceedings of IFIPTM 2008: Joint iTrust and PST Conferences on Privacy, Trust Management and Security,* 103-118.

Princeton University (2010). WordNet - A lexical database for English. Retrieved http://wordnet.princeton.edu/.

Robinson, J.A., & Voronkov, A. (Eds.) (2001). *Handbook of Automated Reasoning*. MIT press

Schafer, J.B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative filtering recommender systems. *The Adaptive Web,* 291-324.

Sebastia, L., Garcia, I., Onaindia, E., & Guzman, C. (2008). e-tourism: A tourist recommendation and planning application. *Tools with Artificial Intelligence, IEEE International Conference on,* 89-96.

Siricharoen, W.V. (2008). Learning semantic web from e-tourism. *Agent and Multi-Agent Systems: Technologies and Applications,* 516-525.

The J. Paul Getty Trust (2010). Art & Architecture Thesaurus Online. Retrieved from http://www.getty.edu/research/conducting_research/vocabularies/aat/

Tammet, T. (1997). Gandalf. *Automated Reasoning, 18*(2), 199-204.

World Wide Web Consortium (W3C) (2010). Rule Interchange Format. Retrieved from http://www.w3.org/2005/rules/wiki/RIF_Working_Group

Acknowledgements

This research has been supported by European Regional Development Fund.

Curriculum Vitae

1. Personal data

Name	Ago Luberg
Date and place of birth	27 March 1983 Tallinn, Estonia
Nationality	Estonian

2. Contact information

Address	Tallinn University of Technology, School of Information Technologies, Department of Software Science, Ehitajate tee 5, 19086 Tallinn, Estonia
Phone	+372 58 666 574
E-mail	ago.luberg@ttu.ee

3. Education

2009–...	Tallinn University of Technology, School of Information Technologies, PhD studies
2001–2007	Tallinn University of Technology, SChool of Information Technologies, Informatics, BSc, MSc

4. Language competence

Estonian	native
English	fluent

5. Professional employment

2020– ...	Tallinn University of Technology, Informatics Bachelor's Program Manager
2017– ...	Tallinn University of Technology, Lecturer
2012–2016	Tallinn University of Technology, Teaching assistant
2011–2016	ELIKO Competence Centre, Senior researcher
2011–2012	Technical University Graz, Marie Curie Research Fellow
2009–2011	ELIKO Competence Centre, Researcher

6. Honours and awards

- 2017, Teacher of the Year, Tallinn University of Technology, School of Information Technologies

7. Defended theses

- 2007, Intelligent pattern-based system and supporting framework for Go-Moku, BSc, supervisor Emeritius Professor Leo Võhandu, Tallinn University of Technology, Institute of Informatics

8. Field of research

- Machine learning
- Information retrieval
- Knowledge discovery

9. Scientific work
Papers

1. T. Tammet, A. Luberg, and P. Järv. Sightsmap: Crowd-sourced popularity of the world places. In L. Cantoni and Z. P. Xiang, editors, *Information and Communication Technologies in Tourism 2013*, pages 314–325, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg

2. A. Luberg, M. Granitzer, H. Wu, P. Järv, and T. Tammet. Information retrieval and deduplication for tourism recommender sightsplanner. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, WIMS '12, New York, NY, USA, 2012. Association for Computing Machinery

3. A. Luberg, P. Järv, and T. Tammet. Information extraction for a tourist recommender system. In M. Fuchs, F. Ricci, and L. Cantoni, editors, *Information and Communication Technologies in Tourism 2012*, pages 332–343, Vienna, 2012. Springer Vienna

4. A. Luberg, P. Järv, K. Schoefegger, and T. Tammet. Context-aware and multilingual information extraction for a tourist recommender system. In *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies*, i-KNOW '11, New York, NY, USA, 2011. Association for Computing Machinery

5. A. Luberg, T. Tammet, and P. Jarv. Extended triple store structure used in recommender system. In *2011 22nd International Workshop on Database and Expert Systems Applications*, pages 539–543, Aug 2011

6. A. Luberg, T. Tammet, and P. Järv. Smart city: A rule-based tourist recommendation system. In R. Law, M. Fuchs, and F. Ricci, editors, *Information and Communication Technologies in Tourism 2011*, pages 51–62, Vienna, 2011. Springer Vienna

7. A. Luberg, J. Pindis, and T. Tammet. Sights, titles and tags: Mining a worldwide photo database for sightseeing. In *Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics*, WIMS 2020, page 149–158, New York, NY, USA, 2020. Association for Computing Machinery

8. A. Kuusik, E. Reilent, I. Lõõbas, and A. Luberg. Data acquisition software architecture for patient monitoring devices. *Elektronika ir Elektrotechnika*, 105(9):97–100, Jan. 2010

9. H. Wu, A. Luberg, and T. Tammet. Ranking domain objects by wisdom of web pages. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, pages 1–4, 2012

10. T. Tammet and A. Luberg. Combining fuzzy and probabilistic reasoning for crowd-sourced categorization and tagging. In *Web Reasoning and Rule Systems: 8th International Conference, RR 2014, Athens, Greece, September 15-17, 2014. Proceedings*, volume 8741, page 247. Springer, 2014

11. I. Lõõbas, E. Reilent, A. Anier, A. Luberg, and A. Kuusik. Towards semantic contextual content-centric assisted living solution. In *The 12th IEEE International Conference on e-Health Networking, Applications and Services*, pages 56–60. IEEE, 2010

Elulookirjeldus

1. Isikuandmed

Nimi	Ago Luberg
Sünniaeg ja -koht	27.03.1983, Tallinn, Eesti
Kodakondsus	Eesti

2. Kontaktandmed

Aadress	Tallinna Tehnikaülikool, Tarkvarateaduse Instituut, Ehitajate tee 5, 19086 Tallinn, Estonia
Telefon	+372 58 666 574
E-post	ago.luberg@ttu.ee

3. Haridus

2009–…	Tallinna Tehnikaülikool, Infotehnoloogia teaduskond, Info- ja kommunikatsioonitehnoloogia, doktoriõpe
2001–2007	Tallinna Tehnikaülikool, Informaatika teaduskond, Informaatika, BSc, MSc

4. Keelteoskus

eesti keel	emakeel
inglise keel	kõrgtase

5. Teenistuskäik

2020– …	Tallinna Tehnikaülikool, Informaatika bakalaureuse programmijuht
2017– …	Tallinna Tehnikaülikool, Lektor
2012–2016	Tallinna Tehnikaülikool, Assistent
2011–2016	ELIKO Tehnoloogia Arenduskeskus OÜ, Vanemteadur
2011–2012	Technical University Graz, Marie Curie Research Fellow
2009–2011	ELIKO Tehnoloogia Arenduskeskus OÜ, Teadur

6. Autasud

- 2017, Aasta õppejõud, Tallinna Tehnikaülikool, Infotehnoloogia teaduskond

7. Kaitstud lõputööd

- 2007, Mustripõhine intelligentne süsteem ja seda toetav paindlik raamistik Go-Moku näitel, BSc, juhendaja emeriitprofessor Leo Võhandu, Tallinna Tehnikaülikool, Informaatikainstituut

8. Teadustöö põhisuunad

- Masinõpe
- Infootsing
- Andmehõive

9. Teadustegevus

Teadusartiklite, konverentsiteeside ja konverentsiettekannete loetelu on toodud ingliskeelse elulookirjelduse juures.

ISSN 2585-6901 (PDF)
ISBN 978-9949-83-692-5 (PDF)