TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Kyrylo Chebotarov 195318IVSM

Implementing a skills recommendation system at MeetFrank

Master's Thesis

Supervisor: Margarita Spitšakova, PhD
Co-supervisor: Priit Järv, PhD

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Kyrylo Chebotarov 195318IVSM

Oskuste soovitamise süsteemi juurutamine MeetFrankis

Magistritöö

Juhendaja: Margarita Spitšakova, PhD

Kaasjuhendaja: Priit Järv, PhD

Tallinn 2021

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Kyrylo Chebotarov

1/05/2021

Abstract

Recruitment has developed dramatically over the last decade due to trends in artificial intelligence, big data, and global sourcing. Nowadays, talents mostly search for work online. On the other side, such digital recruitment platforms' task is to connect job seekers with recruiters in the most accurate and fast way to benefit from it. This study aimed to investigate the utilizing possibilities and their benefits of a skills recommendation system in one of career digital assistants - MeetFrank. Skills recommending is a process of suggesting hard skills to a user based on already mentioned profile attributes or user interactions from the past. This study showed several implementations of skill recommendation systems based on KNN and TF-IDF, Word2Vec, and Naive Bayes models and compared them via skills restoring tasks. The research showed that all three models could work well for skills recommendations, but the Naive Bayes model solves that task the best. It was also proposed the way of data preprocessing steps to obtain meaningful profiles for training data. That included job titles normalization for restoring profile attributes and clustering using hierarchical clustering. Based on the research, it can be assumed that the skills recommendation system is a necessary tool in digital recruitment. This thesis's output was used to improve the talents and recruiters matching algorithm in the MeetFrank, and the proposed recommendation system was approved for further development.

This thesis is written in English and is 40 pages long, including 6 chapters, 9 figures and 3 tables.

Annotatsioon

Värbamine on viimase kümnendi jooksul kiirelt arenenud tehisintellekti, suurandmete ja ülemaailmse hanke mõjul. Tänapäeval otsitakse tööd enamasti veebist. Digitaalsete värbamisplatvormide ülesanne on ühendada tööotsijad värbajatega võimalikult täpselt ja kiirelt, et sellest kasu saada. Lõputöö eesmärk oli uurida digitaalse karjääriassistendi - MeetFranki - oskuste soovitamise süsteemi võimalusi ja nende eeliseid. Oskuste soovitamine on protsess, mille käigus soovitatakse kasutajale profiilidesse tööoskusi olemasolevate profiiliatribuutide või minevikus toimunud kasutajate interaktsioonide põhjal. Lõputöö näitas KNN-i ja TF-IDF-i, Word2Veci ja Naive Bayesi mudelitel põhinevate oskuste soovitamise süsteemide mitmeid rakendusi ja võrdles neid oskuste taastamise ülesannete kaudu. Töös tehtud uuringu alusel võiksid kõik kolm mudelit oskuste soovituste jaoks hästi töötada, kuid Naive Bayesi mudel lahendab selle ülesande kõige paremini. Samuti pakuti välja andmete eeltöötluse sammud treeningandmete jaoks sisukate profiilide saamiseks. See hõlmas ametinimetuste normaliseerimist profiiliatribuutide taastamiseks ja koondamiseks hierarhilise klastrite abil. Uuringu põhjal võib eeldada, et oskuste soovitamise süsteem on digitaalses värbamises oluline töövahend. Selle lõputöö väljundit kasutati talentide ja värbajate sobitamise algoritmi parandamiseks MeetFrankis ning pakutud soovitussüsteem kinnitati edasiseks arendamiseks.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 40 leheküljel, 6 peatükki, 9 joonist, 3 tabelit.

List of abbreviations and terms

BOW	Bag of words
CNN	Convolutional neural network
CPU	Central processing unit
KNN	K-nearest neighbors algorithm
NLP	Natural language processing
PCA	Principal component analysis
QA	Quality assurance
RAM	Random-access memory
RNN	Recurrent neural network
RUS	Random undersampling
SOM	Self-organizing map or Kohonen map
T-SNE	T-distributed stochastic neighbor embedding
TF-IDF	Term frequency - inverse document frequency

Table of contents

List of figures

List of tables

1 Introduction

Nowadays, the recruitment industry is almost fully digitalized. Primarily, it concerns workers from technical specialties. People know that to get a job, they have to create a digital profile (or send a digital resume) to popular local professional platforms. Those platforms' interest is to connect two kinds of their users - job seekers and talent acquirers in the most accurate way and as fast as possible. Since the common ground between recruitment platforms and their users is only one - manual profile creating, it's crucial to provide the best services to the users, to make them able to create the most relevant profiles, because after they are done, the whole responsibility for matching is on the platform. Suppose by mistake the profile was created wrongly or incompletely. In that case, it might affect the quality of the service and reduce the revenue because even though there is a potential pair for matching, the platform just doesn't know about it.

This study explains the value of the skills recommendation system for digital recruitment and the fields of its application based on the example of implementing a skills recommendation system for one of such platforms - MeetFrank [1].

1.1 MeetFrank overview

MeetFrank is a recruitment platform that connects talents with employers. Recruiters are provided with a general overview of real-time market data from a particular speciality (IT, Design, etc.) and hints about the steps they might perform to close the position with the best matching candidate possible. On the other side, talents receive valuable insights and their ranking information about their place within a field they would like to work in. To make the whole ecosystem run, there are many background processes that process data in real-time. This study's general intention is to improve one of such processes - matching talent profiles with job openings.

1.2 Matching talent profiles with job openings

The platform matches employers and talents by calculating the match score. It is a number that defines how well the talent profile fits into the job position. One of the criteria by which the match score is defined is how similar the skills are in the user and job opening profiles. At MeetFrank, the matching algorithm results in a personalized digest feed of job openings in the mobile app for talents. In that feed, job openings are ordered by relevance.

[1] `https://meetfrank.com`

1.3 Skills recommendation system

The main goal of this study is to implement a real-time personalized skills recommendation system at MeetFrank. A recommendation system is a system that aims to predict user preferences for elements that are the content of the system, in our case, hard skills, due to the available user information. The problem that the recommendation system aims to solve is to predict the probability of user U_i has the skill S_k. Having such an instrument might be used in those cases:

- **Scoring** - contextual ordering the subset of skills by relevance to a particular user;
- **Making recommendations** - ranking potentially relevant skills per user by the probability of having those and recommend the top ones to the user.
- Measuring **skills similarity** between a job opening and a talent.

For the recommendation system to work correctly, the initial data on which the recommendations will be based is required. The recommendation system may consume signals from different factors:

1. **Users.** All user's predefined profiles, preferences, and interactions with the system (user profiles, social interractions, groups in which users consist).
2. **Skills.** Characteristics of the predictable content (type of the skill, industry).
3. **Context.** Context of the prediction (speciality, time, relocation, location, feed type, search field).

Basic recommendation systems are mainly categorized into three major forms: *content-filtering*, *collaborative filtering* and hybrid that is a combination of both to resolve the drawback of content and collaborative filtering.

Content-based filtering recommends the content based on the user's predefined profile. In terms of skills recommendation systems, it is the user's skills or preferences for the job he is looking for. A recommendation system makes a decision based only on what the user likes or dislikes.

Collaborative filtering on the other hand, recommends the content based on users' choices with similar tastes. Users who have similar preferences or similar behavior patterns in interacting with a platform will most likely have equivalent recommendations.

The methods for implementing mentioned approaches divides into 2 groups: *memory-based* and *model-based* recommendation systems. The difference between them is that memory-based models do not implement difficult algorithms but instead predict based on huge amounts of historical data when the model-based models are trained machine learning classifiers. The choice of method of forming recommendations also depends on the information that the system will operate.

1.4 Challenges in skills recommendations

With the growth of vocabulary of possible skills, it becomes difficult to recommend the relevant ones. Several challenges separate the skills recommendation system from others domains.

1.4.1 Specialities intersections

Let's consider programming languages. For instance, JavaScript is the universal language widely used in front-end, back-end, embedded, mobile, game, and more other fields of development (Figure 1). If to make the naive assumption that all skills mentioned in user profiles are related, then it would mean that JavaScript has its associated relative skills from different specialties. Consider the following example: the case when we have to predict a "JavaScript back-end developer." In this case, all of those relations from other specialties will contradict each other. The one with the most strong probability win, but if that one is "JavaScript - React," which is from front-end speciality, then the prediction is totally wrong for a back-end developer.

Skill intersections also imply that profiles from completely different specialties might stand really close to each other in vector space. This phenomenon was described in Sections 3.4.2 and 3.4.3. It means that profiles that look almost similar in vector space actually require opposite recommendations.

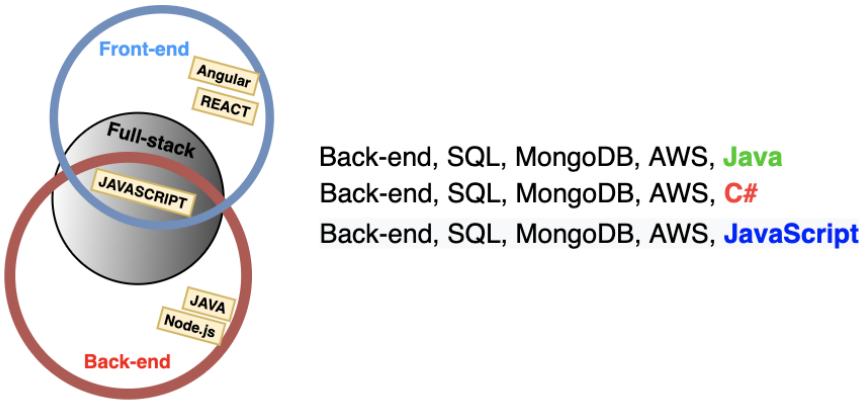


Figure 1. Skill overlapping in different specialties

1.4.2 Data needed for accurate skills predictions

It's crucial to understand that it's impossible to build an accurate skills recommendation system relying only on the user's skillset mentioned in his profile. Consider that the user has three skills: mathematics, physics, and biology, but in his profile, he said

only mathematics and physics. The recommendation system, which takes into account only mentioned skills, will never predict the missing one. Instead, it will recommend something more close to exact sciences.

During this study, it was observed that the quality of recommended skills also depends on the number of skills mentioned in the profile. As shown on the Figure 1, if a user has a profile with the low number of skills mentioned, then there are a huge variety of possible skills to recommend (If the user knows JavaScript, then all JavaScript-based libraries have the same probability of user know them). The same problem appears when a user mentions too many skills from different fields. In both cases, the most popular skill (biased in the training dataset) will be recommended.

Based on this knowledge, it's valid to say that additional information should be taken into account, like the industry, company, country, job title, or other attributes of the user profile for relevant predictions.

1.4.3 Problem of popular skills

There are also 'trendy' or skills that appear almost in each profile. Popular skills might, by mistake, be associated with bad relationships. For example, when a "C# full-stack developer" knows the popular JavaScript, it doesn't mean that JavaScript and C# (that mentioned together in his profile) are related.

1.5 Contributions

This study was mainly focused on analyzing the data provided by the MeetFrank for implementing a skills recommendation system. But during the research process, some side problems were solved. The contributions of this study are as follows:

1. Developed a pipeline for selecting relevant and dropping contradicting skill profiles. (Sections 3.2 to 3.4).
2. Proposed a way of profiles similarity comparison (Section 3.4.3).
3. Proposed a way of restoring skill profile attributes from job titles (Section 3.3).
4. Developed and compared several models of skills recommendation system:
    - KNN Model (Section 4.1).
    - Word embedding based model (Section 4.3).
    - Naive Bayes model (Section 4.2).
    - Results comparison (Section 5.2).
5. Explored and proposed potential ways of applying the skills recommendation system in digital recruitment platforms (Section 5.3).

2 Related work

This section describes the scope of this study and provides an overview of existing solutions.

M. Bastian et al. [1] from the LinkedIn [2]. research team considered the problem of predicting not mentioned skills as discovering latent attributes in social network profiles [16]. Hence, they implemented a skill inference algorithm for suggesting additional skills through a recommendation system as a collaborative filtering solution. They aimed for the system of that kind to be personalized and only to offer the skills members have. They decided that content-based is challenging to apply directly because of the limited availability of information about the skills themselves in profiles. They've chosen collaborative filtering because they noticed that people who have common interests had many common skills (like Apple employees have the skill "Mac OS" in common). Such an approach requires a model that supports a large number of features and a large number of predicted classes. That is why it was used the Naive Bayes classifier where according to Bayes theorem, they proposed to find the probability of "user has a skill," given that "user has profile attribute." After feature selection and comparison by a recall, they came up with industry, company, function, title, group in a feature set. As several correct features were selected, the prediction quality changed from recommending the most frequent skills in the industry to more personalized recommendations. The solution was successfully verified via A/B testing to a large set of LinkedIn members in a system that invited them to complete their profile.

S. Choudhary et al. [3] presented another successful way of using a Naive Bayes classifier as a part of a collaborative recommendation system for job portals. They were recommending jobs to talents using their current skill set extended with similar skills in the global data set that they have acquired. They proved that this approach is highly performant and computationally cheap.

P. Sitikhu et al. [24] performed a comparison between three different approaches for measuring the semantic similarity between two short text documents (cosine similarity using TF-IDF vectors, cosine similarity using word embedding, and soft cosine similarity using word embedding). They empirically proved that TF-IDF with cosine similarities performs the best for such tasks. This paper is relevant for current research, so user job profiles of skills might be represented as short text documents for determining the similarities between skills/profiles.

W. Zhou et al. [29] researched skill gaps in the job market. They decided to try to reduce the skill gaps by accurately identifying the mismatch between the skills expected by employers and those possessed by job seekers. They focused on quantifying the relevance of skills to job titles, relying on just the job posts and skill keywords information. The objective was to return a list of relevant skills for any given job title.

[2] https://www.linkedin.com

Firstly, they grouped all job openings by normalized (via NLP) job titles. Then they calculated the frequency of appearing each of the skills in the group. After, they used TF-IDF to calculate global and local skill uniquenesses (dispersion) to increase the confidence (if a skill is required by a relatively small fraction of job ads with the same title, the skill is not relevant to the title, and if a skill is globally required, then its importance for a particular job title decreases). The solution is successfully utilized for CareerCoach[3] and CareerBuilder[4].

L. Van-Duyet et al. [14] implemented a Skill2vec - neural net system inspired by Word2vec for determining the relations between skills in the recruitment domain. It's highly relevant for this research as they explained the steps and model params of the Word2vec model for reproducing. There are also a lot of works [15, 7] that proved the usage of word embeddings (like word2vec) for solving multi-labeled classification problems. Embedding models usually associate each domain-specific word with a single real-valued vector, representing its different properties. The benefit of this approach is that those embeddings might be extended with CNN, RNN, or word averaging for mapping unknown words to embedding space [17, 22].

[3] https://careercoach.org
[4] https://careerbuilder.com

3 Data

This section describes the data which was used for all experiments in this study. It also explains how data was selected and preprocessed.

Main idea and problems in data preparation. This study's fundamental idea is based on the relationship between different skills, namely closely related skills are more likely to co-occur on the same profile. To achieve this study's goals, it was needed to select as much data as possible while meeting the crucial requirement: *profiles should remain relevant*. In order to create meaningful models, all features mentioned in profiles should not contradict each other but stay consistent. It is a challenging thing to get from the real raw data.

Training models using the raw dataset didn't give any successful results. Here is the list of top reasons that were causing that:

- **Skills don't correspond to selected tracks.** It might happen that users select the "front-end" track, but didn't mention any skill from the front-end, or vice versa.
- **Several profiles in one entity.** Some openings are targeting not one concrete specialist but several at once. Also, users might have very "full-stack" profiles. This leads to mixing the context. Example: "We are looking for C#/Java/Python developers".
- **Mixing context**. It's very important to understand that the skills mentioned in the profile by the user are just things that the user knows or might know. And usually, it happens that users select skills which doesn't make any sense together or even contradict. If to consider the group of users, then some patterns will pop up (Java goes with Spring or C# with .Net), but if to consider each profile separately, then more likely the set of skills will be meaningless.
- **Too many skills.** The more skills mentioned in one profile, the bigger probability that at least several of them are superfluous.
- **Too many tracks.** One particular user might indicate himself with several tracks: front-end development, back-end development, full-stack development, machine learning development, embedded engineering, and QA, but it doesn't mean that all relations with skills in such profile will be applicable to every user.
- **Sparse and biased dataset.** Apart from logical problems, there was a huge problem with the data itself. The current state of the user profiles and job openings at MeetFrank is quite sparse and biased in several specialities/locations (cold start users, poorly filled profiles, jobs only in rare fields). It's possible to observe this problem from Figure 5.

Possibly there are approaches to solve those problems by addressing each of the issues separately based on knowledge of the target domain. However, the size of the dataset allowed to solve those problems by simple filtering irrelevant profiles (profiles which met the criteria mentioned in the list above). It was decided to leave explorations of sophisticated approaches for future work.

Cherry-picking the data. At first sight, filtering irrelevant profiles might seem like cherry-picking data to achieve better results that are not connected with real applications. To understand the validity of this approach, let's consider the classifier for the image recognition task that predicts "if a cat is on the picture." Suppose we remove from the dataset all photos where a cat is taken from the side (we are "cherry-picking" data) to achieve better results. In that case, the classifier will increase its accuracy, but it will never predict correctly in those cases when a cat is taken from aside. However, the situation in this study is a bit different.

In this study, skills recommendations are built on relations between skills (based on occurrences together in profile). If the train data are polluted with the wrong profiles, it will recommend irrelevant skills. The reason for filtering in this study case is to ensure that all relations are healthy (JS goes with CSS and HTML but not with Java or C#). That will build a ground truth closer to reality. Imagine a simple case: collaborative filtering using N nearest neighbors. If the data is polluted (not real skills occurrences in profiles) in train data, it will recommend those incorrectly.

Unlike the example with cat recognition, the final recommender system can recommend relevant skills to both "bad profiles" and "good profiles.

All steps described in this section were performed with consideration to prepare the data and eliminate the problems described in the list above. Section 3.1 describes the high level of data collection, Section 3.2 and Section 3.3 describe the preparing and filtering data based on its features values, Section 3.4 describes the filtering data using clusters and in Section 3.5 is said how the clustering results were used for train-test data splitting.

3.1 Data collection

The real data produced by users of the MeetFrank was used for all research purposes described in this study. The dataset consists of user-profiles and job openings from software engineering speciality created in the period between January 2019 and February 2021. The total numbers of finished job openings and active user profiles before preprocessing were 8437 and 59403 correspondingly. Those numbers have significantly reduced after data cleaning and preprocessing. User profiles and job openings were combined in one dataset with the same number of features and were treated similarly in all further steps. After that, the word "document" will mean either job opening or user profile.

3.2 Data cleaning

Each paragraph below represents the features and the filtering steps that were made based on the value of that feature. The main idea was to drop the obviously irrelevant documents from the provided dataset. From all available features, were selected the ones that have an obvious correlation with hard skills.

Speciality. From 19 available specialities, it was decided to select only software engineering for all experiments in this study due to several reasons:

- Software engineering speciality by itself is a difficult field to work with. Talents and job openings in software engineering have the most diversified variety of possible skills, tracks, and combinations of them.
- After analyzing, it was discovered that data from this speciality would cover all challenges mentioned in Section 1.4.
- This segment is the biggest in MeetFrank at the moment.

Considering the reasoning above, it's valid to assume that if the study results would be successful with data in a software engineering speciality, then all experiments might be applicable and successfully reproduced in other similar domains from data science and management to finance and marketing.

Seniority. The seniority folksonomy consists of such words: Entry, Junior, Mid-level, Senior, Lead, Executive. Users of each seniority level were taken into account. The reason behind that is that seniority only defines the level of experience of the user or candidate, but it doesn't affect the core skill relations. For instance, if a user is a "Junior React.js developer" or "Senior React.js developer," it doesn't change the fact that React.js is strongly related to JavaScript. This effect is also observable from Figure 4, where skills grouped by seniority have almost the same shapes of skill frequencies.

It was noticed that senior people tend to select fewer skills and skills with high entry-level, but junior-level people tend to choose more skills from different areas.

Country. Since MeetFrank is an international company, it has a wide variety of users from different countries. For the current study, porpuses from the provided dataset were selected countries with the most significant markets. The chosen country codes in the descending order of documents count related to them: EE, LT, FI, DE, PK, LV.

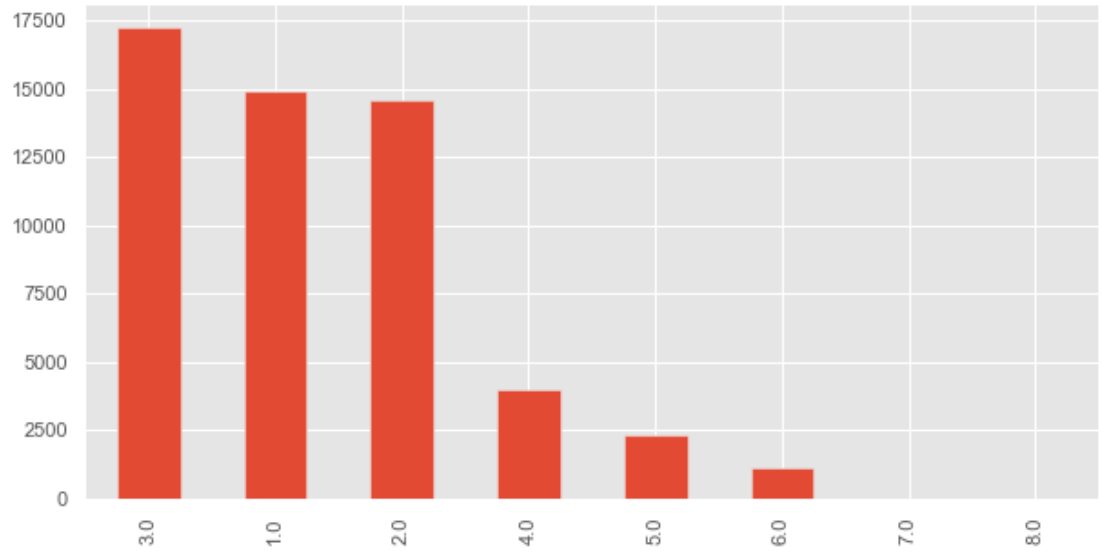


Figure 2. Distribution of number of tracks in profiles

Tracks. There are 17 distinctive tracks available in a software engineering speciality. The distribution of the number of documents with the same track within openings and within user profiles is almost the same. It was decided to drop all profiles which belong to more than 4 tracks or do not belong to anyone. General distribution presented in the Figure 2.

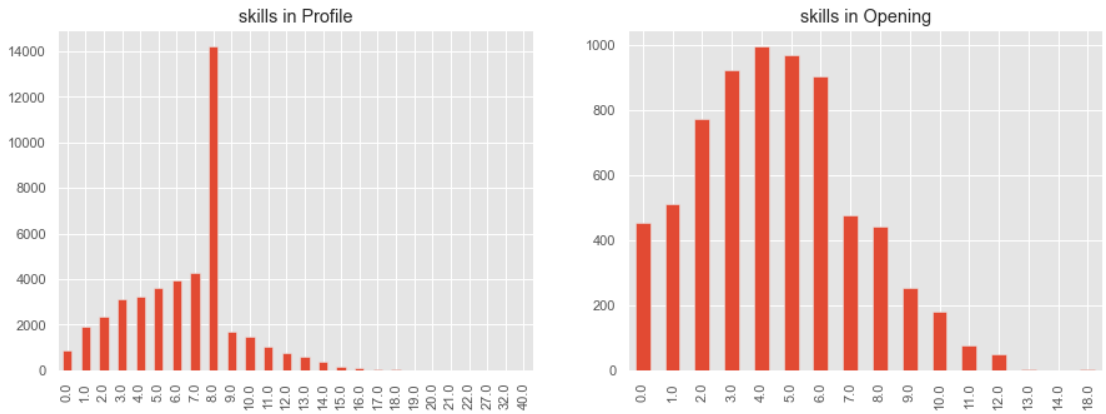


Figure 3. Distribution of the number of skills included in documents

Skills. It was selected 138 unique hard skills related to software engineering speciality. During further preprocessing, this amount was reduced. From Figure 3 it's possible to observe how many skills users tend to put in job openings or user profiles. It was decided

to drop all documents where a number of skills are less than 2 or bigger than 10. Skills, which have not been mentioned in at least 30 documents, were also dropped together with profiles that had those skills. This step might have penalized exotic specialties, but it reduced the long tail of skills for which it was almost impossible to train an accurate predictor. Moreover, skills from the long tail mess with predicting accuracy for other skills, so the overall utility decreases. Also, long-tail skills rarely occur in job offers, so predicting them successfully doesn't help much for the comprehensive recommendation system.

Job titles. Titles of job openings and profile last position titles were taken as they are. It was decided that merge those two in one column would be safe as they represent the same information. The documents which irrelevant job titles were dropped.The process of determining if the job title is irrelevant is described in the section below.

3.3 Data preprocessing

This section describes which additional steps were made in order to make different dataset features understandable for the models.

Job titles. Job title in this context means job opening position title or user's position title. This paragraph explains the value of job titles in hard skills prediction. Even though any prediction model was not build based on job titles, the knowledge gained from this analysis helped in data filtering, data restoring, clusters quality evaluation, and explained some patterns in what information people tend to put into their job titles. Also, these findings might be a powerful foundation for further researches in this area.

After exploring the data, it was noticed that in their job titles, users quite often mention skills or other information useful for skills prediction. Also, users with similar skills tend to put similar job titles, but as described in [8], it doesn't work in the opposite direction - users with similar job titles do not always have identical skills. The average user tends to structure the job title as described in [19, 5].

However, not all users followed the same pattern for job titles. There were a lot of users that put jokes, rude saying, or other irrelevant words or letters into job titles. Also, there was a cluster of 'irrelevant' or 'not experienced' users. For instance, users who mainly work in another speciality but created a software engineering profile, or just switchers from other specialities, students, or people show just finished coding courses. All of them had one characteristic in common. Usually, those profiles were filled randomly, meaning that it was creating unnecessary noise for skills relationships.

It was decided to normalize job titles to gain the ability to filter out irrelevant documents. For that reason, it was developed a pipeline for titles normalization similar to the one that is described in [2]. Eventually, the title normalization using semantic

Raw job title	Result
Jun. dot net automated testing expert	{’seniority’: [’junior’], ’skills’: [’.net’], ’roles’: [’tester’], ’fields’: [’automation’]}
Teamleiter RPA (C++) // Berlin	{’seniority’: [’lead’], ’skills’: [’c++’], ’roles’: [], ’fields’: [’robotic process automation’]}
Ohjelmistokehittäjä, kesätyö (React, node.js)	{’seniority’: [], ’skills’: [’javascript’, ’react.js’, ’node.js’], ’roles’: [’developer’], ’fields’: []}
Mid-level Ruby web devloprs	{’seniority’: [’middle’], ’skills’: [’ruby’], ’roles’: [’developer’], ’fields’: [’web’]}

Table 1. Examples of job titles parsing

similarities seemed to be an overkill, so the normalization stopped on parsing keywords from titles (some examples of parsing in Table 1).

Profiles with titles that didn’t include any word from the expected taxonomy were considered as misleading and dropped.

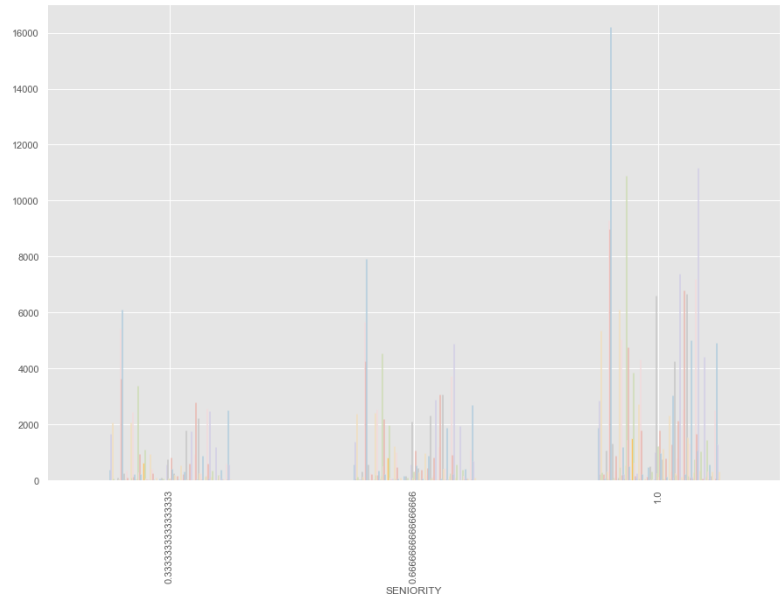


Figure 4. Distribution of skills grouped by seniority weights

Seniority. To make the seniority fature comparable, it was converted to float. There were defined three main seniority groups with corresponding weights:

1. Junior and Entry-level;
2. Mid-level;

3. Senior, lead, executive level.

After weighting, values were normalized to the range [0.33, 1] by dividing each group's weight by 3. The result might be observed from Figure 4.

Restored seniority. 1548 documents didn't have the seniority feature, but 200 of them were restored using a seniority level indicated in the job title using title preprocessing. Others got the label of "Mid-level." Since it relatively not a big part of the data, this step should not break any of the skill relations.

Tracks. Tracks were investigated if some of them are interchangeable, for instance: "QA automation" and "QA testing". Also, it seemed logical to combine profiles in one group, where such tracks were indicated:

- Full-stack development;
- Full-stack development, front-end development, back-end development;
- Front-end development, back-end development.

However, this preprocessing didn't end up with successful results. Also, the fact that all tracks barely correlate hinted at the decision not to do this.

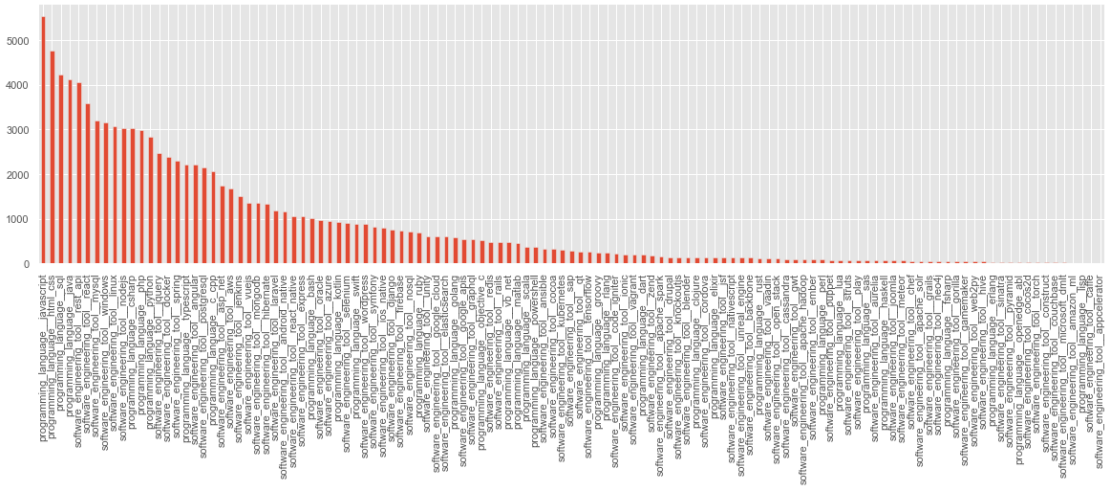


Figure 5. Distribution of skills

Skills and Countries. Skills and countries columns were one-hot encoded. A bar chart of different skill frequencies might be observed in Figure 5. After that, features that

represent skills were transformed using TF-IDF [21]. The reasoning of this is described in Section 3.4.4.

As it was mentioned earlier, the expected result will be to have clusters that represent particular groups of users or subspecialities (subtracks) within a broader software engineering sp

Type. The additional binary column was added, which was determining the type of the document for visualization reasons. It was 0 in the case of user profile and 1 in case of job opening profile.

3.4 Clustering

This is the ice-breaking moment for this research. The subsections below describe the way of thinking and order of insights that led to the final assumptions about the correct approach to data preparation.

After all data manipulations mentioned in previous subsections, the profiles dataset still contained a lot of meaningless documents. The crucial question to answer at this stage was: "*How to select only relevant profiles?*".

One of the possible solutions was to somehow group users by similar characteristics. Then, documents that belong to rare or singleton groups or documents which are far away from group centroids might be considered as noise.

In order to implement this idea, it was decided to first normalize job titles and then group documents by them. This approach didn't provide any satisfying results, but it did verify that this idea will work.

Consequently, it was decided to use another approach, which is well known in the industry for solving such problems - *clustering*. It definitely should be a *hard clustering* approach, because using an overlapping clustering approach [12] doesn't solve the "mixing of the skills" problem. But this choice raised other questions: "*Which clustering algorithm to use?*" and "*By which metric to split clusters?*".

Although high-dimensional data penalize the clustering ability, the average document had around five non-zero features, so clustering should work in this case.

3.4.1 Predefined number of clusters

As it was menteciality (for instance, Full-stack Java Spring developers). But *it's impossible to know in advance how many clusters are there*. Hence, it's obviously an unsupervised learning problem, where the algorithm should try to define how many clusters there are by itself. It would be wrong to force fit the dataset in a predefined number of clusters because it leads to mixing relationships between skills. That's why algorithms like K-Means [10] (algorithms that require a number of clusters as a parameter) will not solve this clustering task.

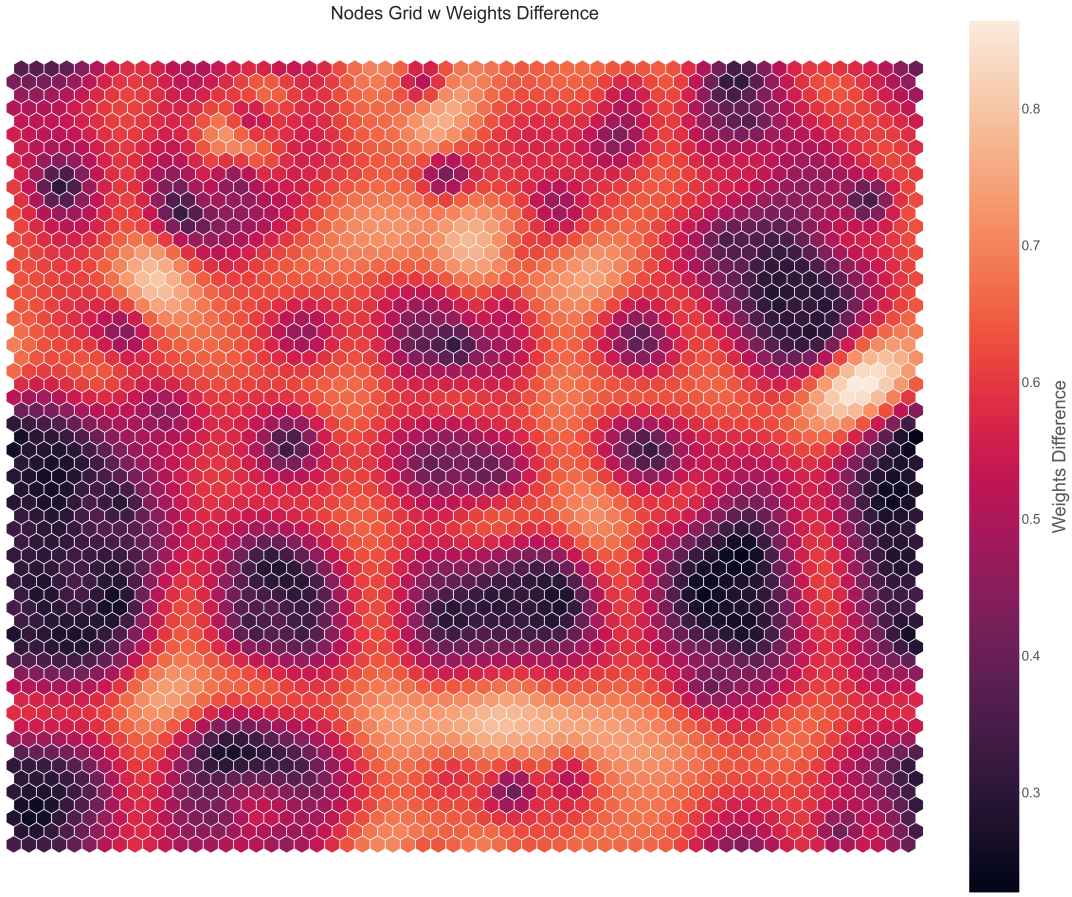


Figure 6. SOM representation of clusters

3.4.2 Density analysis

To understand the distances between potential clusters, it was decided to visualize the data using SOM [13]. It's a powerful tool for data analysis, visualizing, and clustering [4]. Still, in this study, it is used only to understand the density of potential clusters and how imbalanced the clusters are by visualizing the clusters and observing their surrounding borders. Due to its high computational requirements, it was decided to use only part of the dataset (documents related to one country). In [23] it was mentioned that for obtaining meaningful clusters, the grid dimension should be: $5 \cdot \sqrt{N}$, where N is the number of observations. Since the purpose was just to observe the density distribution, it was decided to use 60×60 as grid dimensions.

Dark parts on the SOM differences map (Figure 6) represent different clusters, and the light lines represent the division of the clusters. The resulting map clearly provides such results:

- The clusters do exist, but since the dataset is sparse and biased, the sizes of clusters

vary.

- All clusters are separated, and distances between clusters are quite wide. It consists of documents that do not fully fit into the cluster.
- The width of space between clusters is almost constant. There are no fast transitions from one cluster into another. It's really good to know because it's possible to drop documents that lay on the lines between clusters by selecting points with a constant distance from the cluster's centroids.

Density-based algorithms by the case of DBSCAN [20] also didn't provide any useful clustering results but proved that the dataset is quite consistent. A relatively high *epsilon* parameter (it defines the radius of the circle region in which all points considered to be in the same cluster as the target point) was combining the whole dataset in one cluster. But the classifier with tuned *epsilon* was distributing the great majority of clusters as noise. This experiment provided another useful hint - the *density of data is really high* and different specialities are laying really close to each other, so the existence or absence of only one skill may define if a particular profile is in the group or not. This experiment also proved the presence of the "specialities interceptions" problem described in Section 1.4.1.

3.4.3 Euclidean distance vs cosine similarity

Till this point, all experiments for clustering were conducted using euclidean distance as the main metric. However, results obtained in the experiment described in the previous subsection point out that this metric is completely incorrect. Despite the fact that the euclidean distance between two profile vectors is almost zero (difference in one feature), they might belong to completely different subscpecialities. That's why it was decided to use *cosine similarity* as the main metric for profile comparison.

Nowadays, cosine similarity is a well-known approach for vector comparison. It's widely used in recommendation systems, plagiarism detection, data mining, and even as a loss function for training neural networks. In [25, paragraph 2.4.7] it was mentioned that:

> "Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis. A document can be represented by thousands of attributes..."

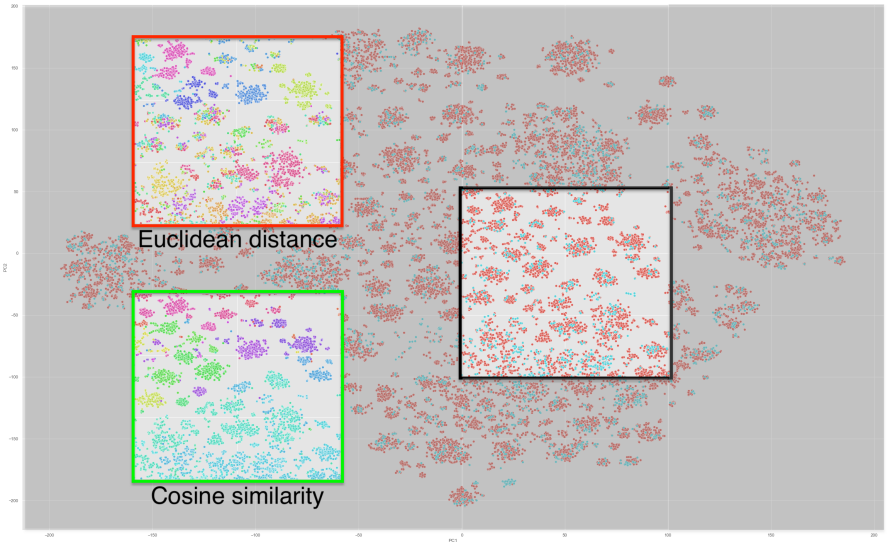


Figure 7. T-SNE visualisation based on cosine similarity

The difference between Euclidean distance and cosine similarity is significant. Form Figure 7 it's possible to see clustering results performed by different methods (blue dots represent job openings, red dots - user profiles). Vectors with small Euclidean distances have a similar "richness" of features, where the existence or absence of one of the features doesn't play a key role. But vectors whose cosine similarity is high look like scaled-up versions of one another. Even though the vectors are one-hot encoded (the vectors' scale should remain constant), cosine similarities are more sensitive to the presence of features.

Another benefit of using cosine similarity is that its result is already normalized and bound by a constrained range of 0 and 1.

3.4.4 TF-IDF transformation

Using cosine similarities as a clustering metric significantly improved the quality of obtained clusters, but there was one obvious and yet unsolved problem: the skill vectors were one-hot encoded, meaning all skills in profile had the same weight. This problem was solved using TF-IDF [11, paragraph 6.5] transformation in combination with L2 normalization. After the transformation, non-zero values were transformed into TF-IDF weights, where bigger values imply a strong relationship with the document they appear in. For example, the value of JavaScript in combination "JavaScript, C#, .NET" will be lower than in combination "JavaScript, React, Node.js".

3.4.5 Visualization

Initially, the PCA [9] was considered as the main tool for cluster visualization because of valuing the computation time and dataset size. However, PCA was showing results where outliers and exotic specialties were extremely penalized (the great majority of skill features is very sparse, which can be seen from Figure 5). Hence, it was decided to use t-SNE [26] instead, which can handle outliers. By running this model with different *perplexity* parameter values and observing similar patterns, it was verified that this method is applicable. All model params were tuned using [28]. It's important to notice that according to reasoning from Section 3.4.3, the *metric* parameters was set to *cosine*.

3.4.6 Obtaining clusters

After all experiments, the final decision for splitting documents into groups was to use a hierarchical agglomerative clustering algorithm [18] such as it fulfills all requirements. For clustering were selected only tracks and skills features. Average linkage was performed using cosine similarity as a metric. Results are available in Appendix 2. There are several ways of how to obtain clusters from the existing linkage. In this study, it was decided to consider these methods:

- **By distance between nodes.** Clusters are forming so that all documents from one cluster have no greater distance than asked.
- **By a number of clusters.** Algorithm finds a minimum distance between two documents in the same cluster and groups documents in the way that the final number of formed clusters is not greater than asked.
- **By inconsistency of clusters.** The inconsistency parameter is provided. If a cluster node and all its descendants have an inconsistent value less than or equal to that parameter, all its leaf descendants belong to the same cluster.

To select the best clustering approach, the quality of obtained clusters after each experiment were evaluated by such qualifications (ordered by priority):

1. **The distribution of tracks and skills in one cluster**. If in one cluster the frequency distribution of tracks (skills) was flat (a lot of different tracks/skills without obvious leaders) or if the combination of contradicting tracks (skills) was observed, then the clustering approach was considered as bad.
2. **The number of obtained clusters and the average size of the cluster.** If the number of clusters was too high or the average size of clusters was low, the clustering approach was considered bad.

3. **How well the data is fitted into the clusters**. For this purpose, the Silhouette score [27] was used. The comparison results are available in Appendix 3.

4. **The similarity of job titles within one cluster.**

Results. At the end, the results provided by the 'distance' hierarchical clustering method were the most accurate. All noise documents were filtered. This approach dropped documents that belonged to small groups (groups with a size of less than a threshold) or documents that stand too far from cluster centroids. The t-SNE visualization might be observed in Appendix 4. The clustering results might be observed in Appendix 5.

3.5 Train-test splitting

Firstly, the random split with a ratio of 80/20 was used for splitting data into test and train sets. However, there was an issue with that approach. Since the data is unbalanced, in most cases, the documents from minor subspecialties were entirely present only in the validation set. It was confusing recommendations.

That problem was eliminated after performing clustering. It became logical to split data using clusters information via stratified k-fold. It ensures that documents from each subspeciality (cluster) exist in the train and test set in asked ratio.

4 Models

This section explains approaches that were conducted to implement the hard skills recommendation system. This study's main goal was to reproduce the success already achieved by other researchers in the same domain. The requirements by which the solutions were selected and compared are following:

- Scalability. The solutions that perform similarly well on the train data with a different number of documents or features). The response speed and the time needed to train/retrain/update the model were also considered.
- Computation power requirements. Complex structured solutions might require more RAM/CPU usage, bringing additional costs for deploying the system. If relatively same results were achieved with the more straightforward approach, that one would be preferable.
- Accuracy. The solutions should perform well in terms of accuracy.

After reviewing related work (Section 2), the thee methods were emphasized that feet into all requirements:

1. TF-IDF transformation with a dedicated metric for measuring the similarities.
2. Word embedding based model.
3. Naive Bayes classification model.

The following subsections provide only the implementation steps (training the model) for all of those models. They aimed to solve the same multilabel classification task (Section 5.1) of predicting skills using the same data (identical train/test split).

4.1 KNN model

After analyzing the statistics of obtained clusters (Appendix 5), it was decided that even a collaborative filtering approach using a simple KNN model should provide realistic skills recommendations. The classifier was designed as follows:

1. Find N the closest documents to the target document;
2. Replace all skills that are already present in the target profile with 0 to avoid recommendation of already mentioned skills.
3. Compute the mean of the rest of skills mentioned in closest documents;
4. Return the skills sorted in descending order.

There might be situations when the target document already has all skills which were mentioned in the closest N documents. This problem was solved by empirically defining N as 150 to have at least one new skill.

Data preparation. For this method, it was possible to use raw data obtained after clustering. TF-IDF transformer was fitted on the train set, but both validation and train sets were transformed. The best performance was achieved using such features: skills, tracks, country.

Performance optimization. This approach doesn't require any additional steps for training the model, but there were made a performance optimization. To avoid recalculations during cross validation, it was decided to precompute the similarities matrix for the whole dataset using cosine similarity as a metric.

4.2 Naive Bayes model

Data preparation. To implement the multilabel classifier using the Naive Bayes model, it was needed to make some additional data preparation. The model requires an input train data in the format "features" and "labels." Since the model is predicting skills (skills are "labels") based on skills (skills are also "features"), it was necessary to ensure that train features don't include the label (feature with 100% correlation). It was decided to preprocess the data in a specific way (Figure 8). If the document has N skills, then that document was transformed into N rows with corresponding labels, but in the "features," that "label" skill was set to 0.

There is a severe skew in the label distribution. Generally, it is solved by methods that randomly or in a sophisticated way drop or produce additional data points (undersampling and oversampling). The imbalance in labels (Figure 5) was partially fixed using undersampling technic - Tomek links [6], as it provided better results compared to RUS. No one of oversampling approaches for minor classes ended up successfully.

The best performance was achieved using such features: skills, tracks, country.

Skill features			Other features		
JavaScript	HTML	CSS	Country_EE	Country_DE	...
0.5	0.6	0,7	1	0	...

Other features	Skill features			
	JavaScript	HTML	CSS	Label
...	0	0.6	0.7	JavaScript
...	0.5	0	0.7	HTML
...	0.5	0.6	0	CSS

Figure 8. Example of document tranformation to obtain labels and features

4.3 Word embedding based model

For implementing the word embedding model, it was decided to use the Word2Vec model implemented by Gensim [5].

Data preparation. Unlike the two previous models, it was impossible to use any other features but skills and tracks. All documents from the train set were transformed into BOW, but documents from the validation set were transformed as described in 4.2.

Parameters tuning. Word2Vec model is very sensitive to parameters. One incorrectly selected parameter might make the whole model accuracy to 0. Here are the empirically chosen via cross-validation parameters that worked the best for the existing dataset:

- **Window parameter.** Since Word2Vec is sensitive to the order of words in the document, so doesn't treat skill features independently. It considers words as related if they appear in the documents in the 'window' length from each other. To eliminate this effect, it was decided to choose the maximum document skills length as a window parameter.
- **Size parameter.** This parameter defines the dimensionality of the word vectors. To stay accurate with an extremely low number of features (less than 200), this parameter should also be low. It was set to 6.
- **Sg parameter.** This parameter defines the training algorithm. It was set to 1 to use skip-gram.
- **Restrict_vocab parameter.** To make this parameter work, the skills vocabulary should be sorted by descending frequency (**sorted_vocab** set to 1). Then, during the prediction, this parameter defines how many skills to check from the top in the vocabulary order. It was set to 50.

Prediction. The skills recommendation using the Word2Vec model happens via finding the most similar skill terms from the pre-trained vector space using *model.wv.most_similar* method.

[5] `https://radimrehurek.com/gensim/models/word2vec.html`

5 Analysis of the results

5.1 Results evaluation

For understanding which of the selected models performs the best, the models were asked to solve the same multilabel classification tasks and then evaluated via:

1. Evaluation of performance metrics of **skills restoring**.
2. **Expert evaluation of predicted skills** to the same user profiles.

Skills restoring. To test skills restoring, it was selected a vocabulary of 30 skills (top skills ordered by frequency). One by one, each skill from each document (job opening or user profile) from the validation set was hidden (replaced with 0) on purpose. Then, the rest of such document was given to the model. If the model responds with the expected skill (previously hidden), then such entry was considered true positive (this process is shown in Figure 8).

It's essential to understand that it's impossible to get 100% accuracy in such kinds of tasks because of various reasons (some of them are mentioned in Section 1.4). In other words, a recommended skill might be correct (fits into the profile as a good recommendation), but it doesn't mean that exactly that skill was mentioned in the profile before. So, even if the system provides valid recommendations, the model's accuracy using this evaluation method remains low.

Recommendation position	Metric	KNN	Naive Bayes	Word2Vec
K = 1	Accuracy	0.45	0.41	0.26
	Recall	0.45	0.41	0.26
	Precesion	0.45	0.44	0.43
	F1 score	0.44	0.42	0.27
K = 2	Accuracy	0.15	0.15	0.15
	Recall	0.15	0.15	0.15
	Precesion	0.17	0.21	0.26
	F1 score	0.15	0.16	0.16

Table 2. Performance metrics of skills restoring

The ability to restore mentioned skills doesn't answer the question "how good the model recommends skills," but it answers "how good the model restore skills" or "how well the model fitted the data." However, it's valid to assume that the ability to restore skill correlates with making good recommendations. The model that converts the best makes the best skill recommendations.

For evaluation, such metrics were chosen: accuracy, recall, precision, and f1 score at K position of recommended skills list, since it's a multilabel classification task. The models were tuned in order to achieve the biggest recall possible since it's better not to recommend skill at all than recommend irrelevant skill. Still, since the dataset is unbalanced, it's crucial to take into account other metrics also. Those metrics were calculated per skill from the target set. Then, to evaluate the whole model's performance, each metric's weighted average score was calculated correspondingly.

The results for comparison might be observed in Table 2. It's possible to notice (based on K) that predictions are sorted by relevance in descending order. More detailed statistics (confusion matrices and distributions of metric values per skill for each model) might be found in Appendix 6.

Original profile	**KNN**	**Naive Bayes**	**Word2Vec**
Tracks: Back-end development Front-end development Full-stack development Skills: MongoDB React Node.js	1) Javascript 2) HTML&CSS 3) Express.js 4) Rest API 5) Angular.js 6) Vue.js 7) AWS 8) PostgreSQL 9) Typescript 10) MySQL	1) Express.js 2) Javascript 3) HTML&CSS 4) Typescript 5) Rest API 6) React Native 7) Angular.js 8) Vue.js 9) JQuery 10) PostgreSQL	1) Javascript 2) Typescript 3) Express.js 4) Vue.js 5) Angular.js 6) HTML&CSS 7) Rest API 8) React Native 9) JQuery 10) PostgreSQL
Tracks: Back-end development Full-stack development Skills: Java Linux Windows	1) Spring 2) SQL 3) Rest API 4) HML&CSS 5) Javascript 6) Hibernate 7) MySQL 8)PostgreSQL 9)Python 10) React.js	1) Spring 2) SQL 3) Hibernate 4) MySQL 5) Python 6) Docker 7) PostgreSQL 8) C/C++ 9) Rest API 10) Jenkins	1) SQL 2) Rest API 3) Docker 4) PostgreSQL 5) AWS 6) MySQL 7) Spring 8) Hibernate 9) Javascript 10) Jenkins

Table 3. Example of predictions made to the same profiles

Expert evaluation of predicted skills. After all three models were implemented, it became possible to evaluate their performance by comparing the predictions made by each of them to the same profile. The vocabulary for such predictions was limited on purpose, to see almost the same skills predictions but in a different order. Since the skills predicted by the models are sorted by relevancy, the limitation of predicted vocabulary

made it easier to compare the results. If the skill is relevant and it's at the top of the list, then such prediction was considered correct. The example of such predictions is shown in Table 3.

5.2 Results comparison

Confusion matrix analyses. It's possible to observe from confusion matrices (Appendix 6) that all of three methods have different accuracy, but they all have the same "problem." On average, they confuse interchangeable skills. Suppose React.js, Vue.js, or Angular.js. All of them are relevant to any front-end/full-stack developer who knows JavaScript. Or skills like "Rest API," which applies to almost every development field.

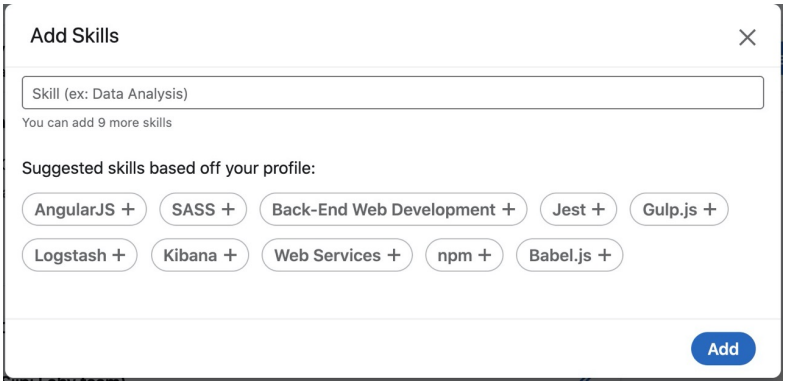


Figure 9. Recommendations of skills in LinkedIn

The accuracy is low for restoring skills, but the shown confusion is a good indicator of these models' role in the skills recommendation system. In the restoring skills experiment, only one skill from the whole recommended list was used. But for skills recommendations, it's possible to suggest several skills at once. As told in their paper [1], LinkedIn provides the user with already ten recommendations (Figure 9). They compared using proved that such an approach is most acceptable by users. In this case, the user's probability has at least one from 10 recommendations is incredibly high.

Models comparison. Even though the KNN model showed the best accuracy, the Naive Bayes model was chosen as the best approach from selected methods for the skills recommendation system. All methods have shown pretty similar and satisfactory results, but each of them has its downsides.

The benefit of using the Word2Vec model is that it supports finding recommendations using positive and negative similarities. For instance, it's possible to include Java and "Full-stack" speciality and exclude "Front-end" speciality. Theoretically, the model will recommend only skills related to the "Back-end" speciality. However, the downside is

that to train meaningful embeddings. The model requires many data (not thousands of documents but hundreds of thousands). Not having enough data makes it much more challenging to make good predictions.

The raw Word2Vec implementation doesn't support features except skills and tracks. It means that it's difficult to extend the model by providing additional signals. This was the main reason for not going with the Word2Vec model, because it just cannot consume all available information.

The KNN model's downside is that it's very data dependant. The predicted results rely only on the train documents, and if data selected not properly, the recommendations for some clusters of users might be off. Such an approach is very biased to the training data skills. However, it's a perfect choice when data is not sparse/biased, or it might be a good choice for other domains that are not so diversified).

The training process using *memory-based*(KNN) approach is also complicated because it requires a lot of work if the training dataset needs to be updated. Of course, there are libraries or plugins to databases that help in managing data or finding similarities faster (like "Annoy"[6] that is working in Spotify[7] for such purposes). But it's much easier to support *model-based* solutions instead of *memory-based*. First of all *memory-based* solution might be trained once, and it's good to use until the new labels/features needed to be introduced. The model itself is lightweight, and it stores only serialized weights, while *memory-based* solutions require all data to be stored somehow. That is why the Nayive Bayes model was selected as the best one. In addition to this, the recommendations made by it were the most relevant starting from the top of the list comparing to other models.

5.3 Applications

5.3.1 Clustering usage

A deeper analysis of achieved clusters in Section 3.4 proved that clustering is a very powerful tool in the recruitement domain. It helps in:

1. Understanding the difference between supply/demand.
2. Detecting exotic/popular subspecialities.
3. Understand which tools/jobs are trendy in which country/speciality.
4. Understanding which communities/groups of talents are possible to create.
5. Using groups of people for push strategies.
6. Detecting better target groups for A/B testing.

[6] `https://github.com/spotify/annoy`
[7] `https://www.spotify.com/`

7. Developing functionality based on groups of people like jobs feed composition or caching strategies.

8. Better targeting and choosing the marketing strategies.

5.3.2 Skills recommendation system usage

There are also a lot of areas where a skills recommendation system might be used:

1. **Improve the talents and job openings profiles matching.** Sometimes, either recruiters or talents don't fully specify all their skills/want to have in the profiles. As a result, they don't use the system's whole opportunities since the matching algorithm doesn't have enough information to join job openings with talent. It's possible to extend the matching algorithm by using newly predicted hidden skill vectors instead of calculating match scores using raw profiles.

2. **Adding new skills to the platform.** Within the great majority of digital recruitment platforms, the set of skills operated in the system is predefined and limited. Even having a vast vocabulary of skill does not cover all possible scenarios. If users are not able to add user-defined skills, so the system should do it. However, merely introducing a new skill into the platform is inefficient. First talents/job openings which will use their skill slot for newly added skill will be affected by not working matching algorithm. Since matching algorithm relies on having the same set of skills from both sides: user profile and job opening. Users who just picked a newly added skill will not be matched because there are missing job openings, including new skills. Some time should pass to make the freshly added skill work as expected.

   The having of skills recommendation system may solve this problem by adding new skills to the platform in the way that they are ready to be used by matching algorithm from day one.

3. **Measuring skills similarity.** A talent is likely to show interest in a job that has a skill profile similar to their own skill set.

4. **Improve UI/UX**. Having dynamic context suggestions in search fields and during the profile creation stage might be very convenient for the user. That might result in creating better quality profiles.

5. **Balancing the supply/demand.** It's possible to find the gap between skills defined in job openings and talent profiles. Then recommend to users skills from the gap. This will eliminate cases when the system doesn't know about potentially suitable matches.

6. **Push strategy.** Skills recommendations might be used as push notifications to increase engagement. Also, it's a known fact that Profiles don't receive frequent updates. The great majority of users don't tend to update their profiles. Mostly, they leave them as they are after creation.

5.4 Future work

5.4.1 External data sources

All experiments in this study were conducted using data provided by MeetFrank. However, that data is highly biased and sparse. It's possible to (partially) mix train data with data gained from other sources to fix the imbalanced features issue. Using additional sources might provide skills that are yet not known in MeetFrank.

5.4.2 Better data preprocessing

The performance of the recommendation system fully depends on the ground truth of the train data. Unfortunately, even after preprocessing described in Section 3.3, a lot of inconsistent profiles remained. The possible solutions are:

1. After obtaining clusters, it's possible to drop minor skills from all profiles in the cluster. For instance, if there is a cluster of 200 users and there are skills that mentioned only once, it's possible to drop those skill because more likely they are outliners.

2. It makes sense to extend (or eliminate this situation) when the user indicated a profile that he belongs to the track, be he didn't mention any skills from that track (or vice versa).

3. It's possible to split generic profiles into several profiles. It means to split profiles with several tracks or openings with several positions in one profile. For instance, when all five programming languages and ten tools are mentioned in one profile, it doesn't mean that all of those tools are related to all five languages.

5.4.3 Disambiguation

As mentioned in [1], it makes sense to treat users from different clusters differently to provide more relevant contextual recommendations. For example, the skill "AWS" has a separate meaning in the contexts of "Back-end Development" speciality and "DevOps Engineering" speciality, even though it's valid in both cases.

5.4.4 Practical application

The output of this thesis was used for improvements in the MeetFrank. The existing talent profile and job opening matching algorithm for software engineering speciality was replaced with an approach introduced in Section 4.1. It was decided to use the value of the cosine of the angle between two transformed via TF-IDF vectors as a match score. It improved the behavior of a "digest feed," in which users might found relevant job offers based on a wide range of factors: speciality, location, relocation preferences, salary preferences, and skills match score. Also, it introduced a new feature - "top picks," that provides users with the best match job offers based on their skillset.

The implementation of the skills recommendation system was approved in further steps.

6 Summary

At this point, this study has proved that the skills recommendation system plays a crucial role in the digital recruitment domain by investigating possible implementations of such a system for MeetFrank. This study has demonstrated a wide range of areas where a skills recommendation system can be utilized, starting from improving UI/UX and engagement level for the end-users and ending with enhancing the core logic of digital recruitment platform - matching job openings with talent profiles.

This study has suggested a pipeline for initial data preprocessing to collect the most relevant user profiles to obtain the train set where ground truth skills relationships are close to reality. One of the main steps in that pipeline was clustering, which also provides many profitable utilizations.

Finally, three models for skills recommendations were implemented and compared: KNN model based on cosine similarity and TF-IDF, Naive Bayes model, and Word2Vec model. Even though the KNN model showed the best performance metrics during skills restoring tasks, it was decided that the Naive Bayes Bayes is a more viable solution for the skills recommendation task.

References

[1] Mathieu Bastian, Matthew Hayes, William Vaughan, Sam Shah, Peter Skomoroch, Hyungjin Kim, Sal Uryasev, and Christopher Lloyd. Linkedin skills: Large-scale topic extraction and inference. 10 2014.

[2] Ron Bekkerman and Matan Gavish. High-precision phrase-based document classification on a modern scale. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, page 231–239, New York, NY, USA, 2011. Association for Computing Machinery.

[3] Savita Choudhary, Siddanth Koul, Shridhar Mishra, Anunay Thakur, and Rishabh Jain. Collaborative job prediction based on naïve bayes classifier using python platform. pages 302–306, 10 2016.

[4] Marie Cottrell, Madalina Olteanu, Fabrice Rossi, and Nathalie Villa-Vialaneix. Self-organizing maps, theory and applications. *Revista de Investigacion Operacional*, 39(1):1–22, 2018.

[5] Ross Edward. What's in a Job Ad Title. `https://skeptric.com/job-title/`. Accessed: 2021-02-23.

[6] T Elhassan and M Aljurf. Classification of imbalance data using tomek link (t-link) combined with random under-sampling (rus) as a data reduction method. 2016.

[7] Jiechieu Florentin Flambeau and Norbert Tsopze. Skills prediction based on multi-label resume classification using cnn with model predictions explanation. *Neural Computing and Applications*, 08 2020.

[8] Lee Hung. The problem with job titles. `https://www.linkedin.com/pulse/problem-job-titles-part-2-hung-lee/`. Accessed: 2021-02-23.

[9] Dong Hyun Jeong, Caroline Jeong, William Ziemkiewicz, Remco Ribarsky, and Chang. Understanding principal component analysis using a visual analytics tool. 01 2010.

[10] Xin Jin and Jiawei Han. *K-Means Clustering*, pages 563–564. Springer US, Boston, MA, 2010.

[11] Dan Jurafsky and James H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Pearson Prentice Hall, Upper Saddle River, N.J., 2009.

[12] Atefeh Khazaei, Mohammad Ghasemzadeh, and Dieter Gollmann. Overlapping clustering for textual data. In *Proceedings of the 2018 7th International Conference on Software and Computer Applications*, ICSCA 2018, page 115–118, New York, NY, USA, 2018. Association for Computing Machinery.

[13] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[14] Van-Duyet Le, Vo Minh Quan, and Dang Quang An. Skill2vec: Machine learning approaches for determining the relevant skill from job description. *CoRR*, abs/1707.09751, 2017.

[15] Ladislav Lenc and Pavel Král. Word embeddings for multi-label document classification. pages 431–437, 11 2017.

[16] Jiwei Li, Alan Ritter, and Eduard Hovy. Weakly supervised user profile extraction from twitter. volume 1, pages 165–174, 06 2014.

[17] Pranava Swaroop Madhyastha, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Mapping unseen words to task-trained embedding spaces, 2016.

[18] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms, 2011.

[19] ongig.com. Job Titles: The Definitive Guide. `https://www.ongig.com/job-titles#/#hierarchy`. Accessed: 2021-02-23.

[20] Anant Ram, Jalal Sunita, Anand Jalal, and Kumar Manoj. A density based algorithm for discovering density varied clusters in large spatial databases. *International Journal of Computer Applications*, 3, 06 2010.

[21] Juan Ramos. Using tf-idf to determine word relevance in document queries. 01 2003.

[22] Nair Sarath. Extend multi-label classification to map unknown words to embedding space. `https://medium.com/@s4sarath/extend-multi-label-classification-to-map-unknown-words-to-embedding-space-f839bfe2b622`. Accessed: 2021-02-23.

[23] Andrii Shalaginov and Katrin Franke. A new method for an optimal som size determination in neuro-fuzzy for the digital forensics applications. pages 549–563, 06 2015.

[24] Pinky Sitikhu, Kritish Pahi, Pujan Thapa, and Subarna Shakya. A comparison of semantic similarity methods for maximum human interpretability. *CoRR*, abs/1910.09129, 2019.

[25] G. Thamaraiselvi and A. Kaliammal. Data mining: Concepts and techniques. *SRELS Journal of Information Management*, 41(4), 2014.

[26] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[27] Fei Wang, Hector-Hugo Franco-Penya, John D. Kelleher, John Pugh, and Robert Ross. An analysis of the application of simplified silhouette to the evaluation of k-means clustering validity. In Petra Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, pages 291–305, Cham, 2017. Springer International Publishing.

[28] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. *Distill*, 2016.

[29] Wenjun Zhou, Yun Zhu, Faizan Javed, Mahmudur Rahman, Janani Balaji, and Matt McNair. Quantifying skill relevance to job titles. pages 1532–1541, 12 2016.

Appendix 1 – Licence

Non-exclusive licence for reproduction and publication of a graduation thesis

I, Kyrylo Chebotarov,

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Implementing a skills recommendation system at MeetFrank", supervised by Margarita Spitšakova and Priit Järv

    1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

    1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.

3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

1/05/2021

Appendix 2 – Hierarchical clustering dendrograms

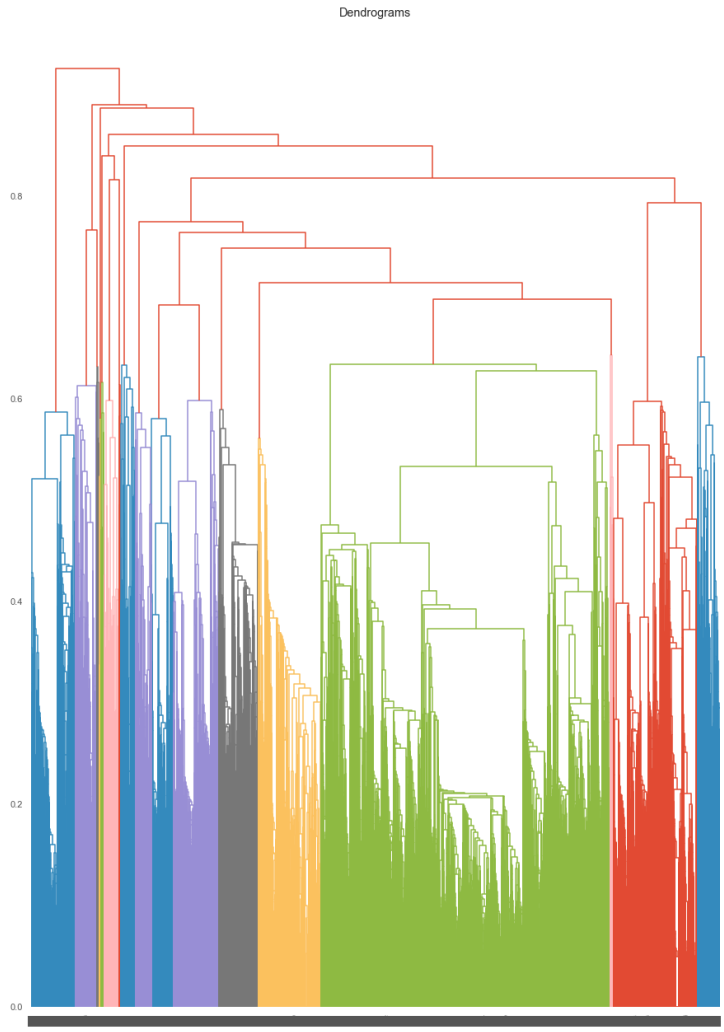


Figure 10. Dendrograms that shows the hierarchical relationship between documents

Appendix 3 – Hierarchical clustering methods comparison

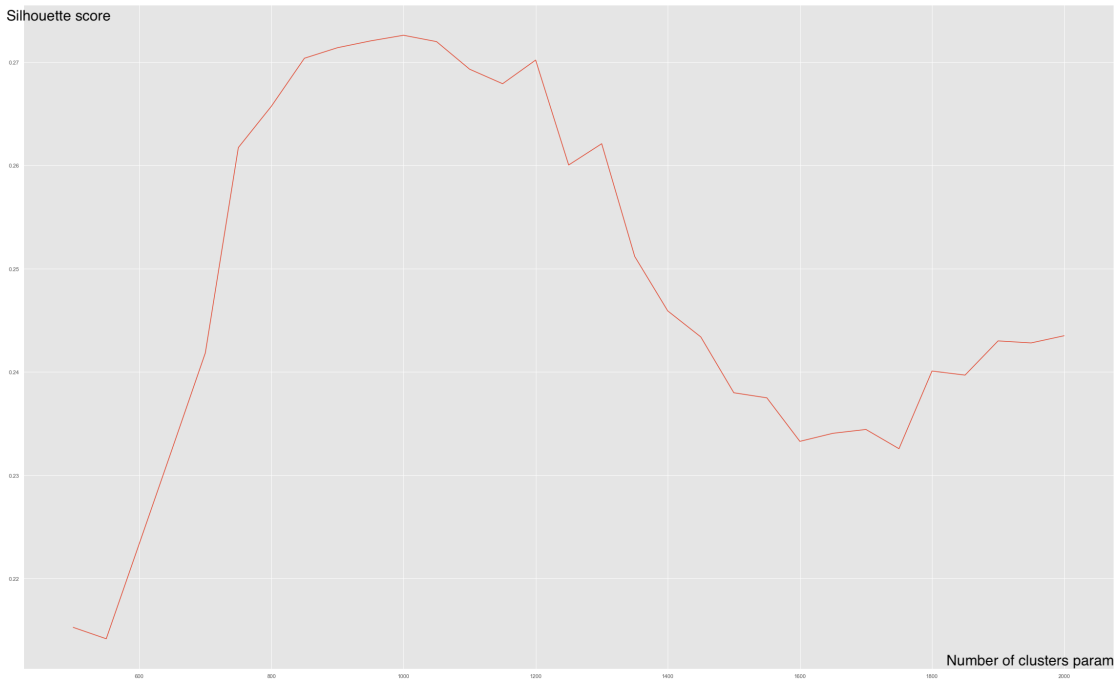


Figure 11. Silhouette score distribution of hierarchical clustering using "maxclast" method

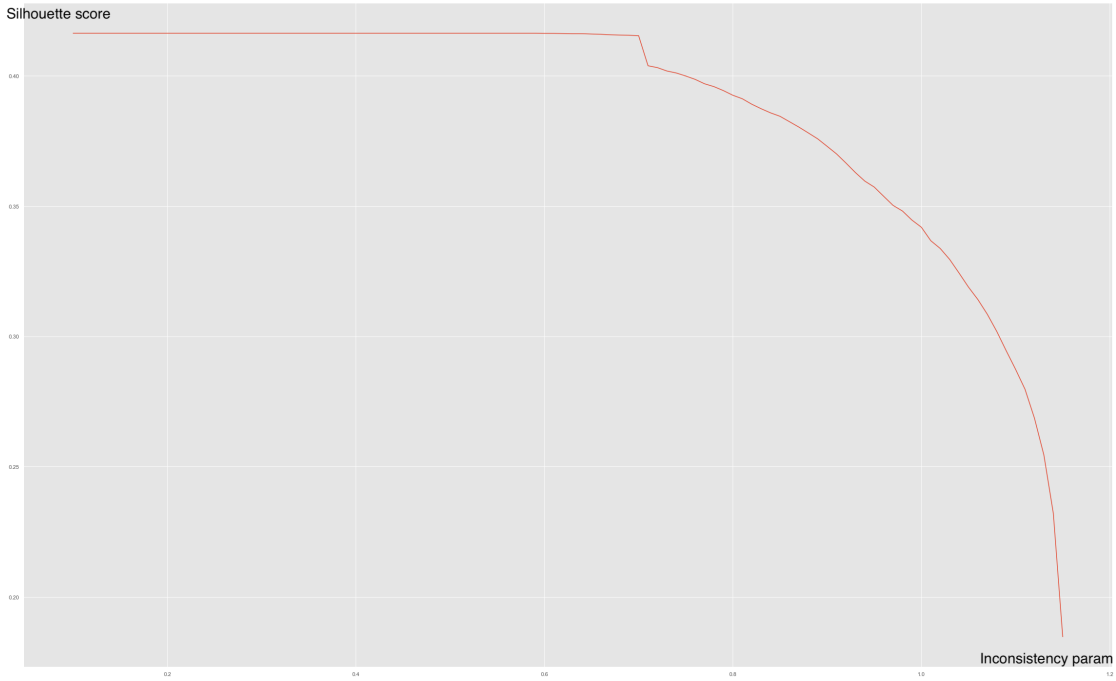


Figure 12. Silhouette score distribution of hierarchical clustering using "inconsistency" method

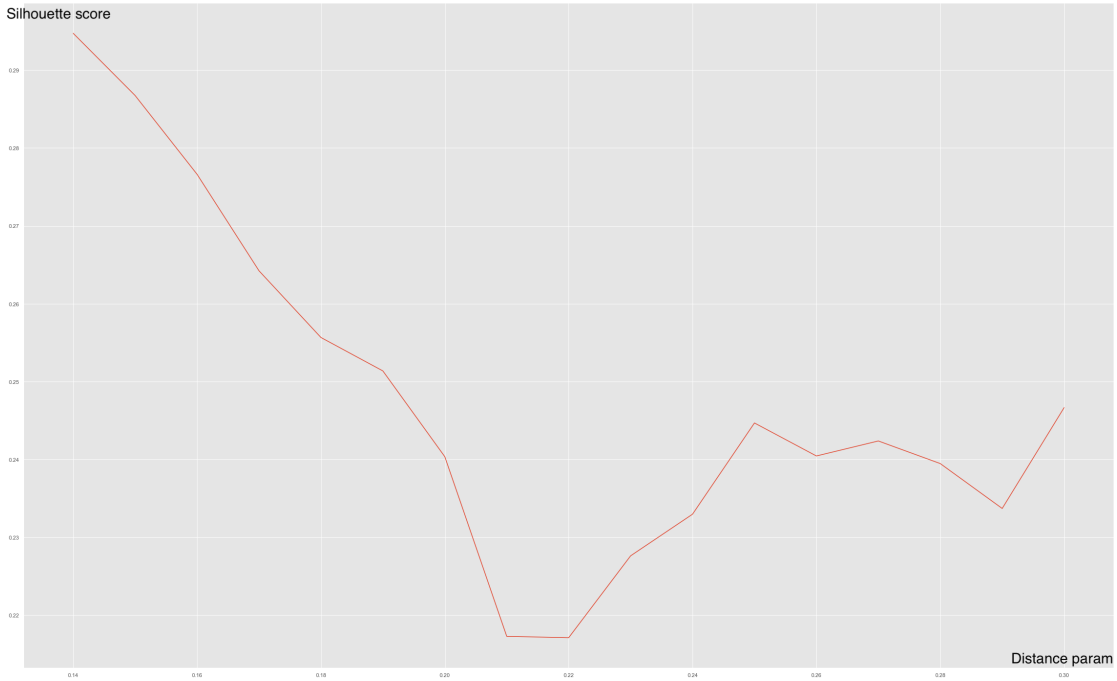
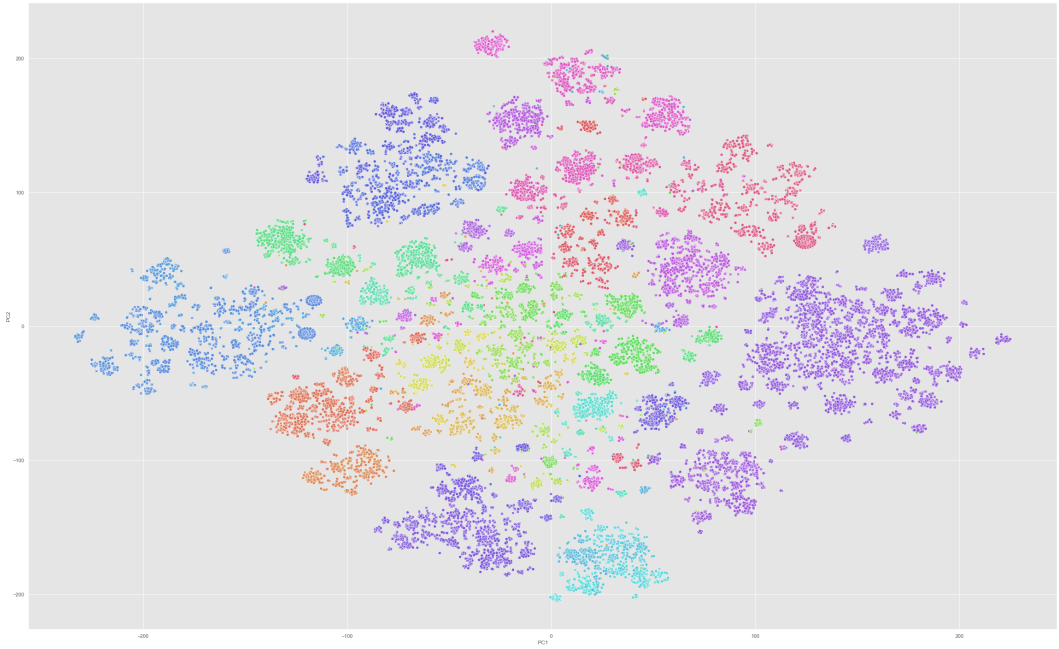


Figure 13. Silhouette score distribution of hierarchical clustering using "distance" method

Appendix 4 – Noise filtering using hierarchical clustering

Before:



After:

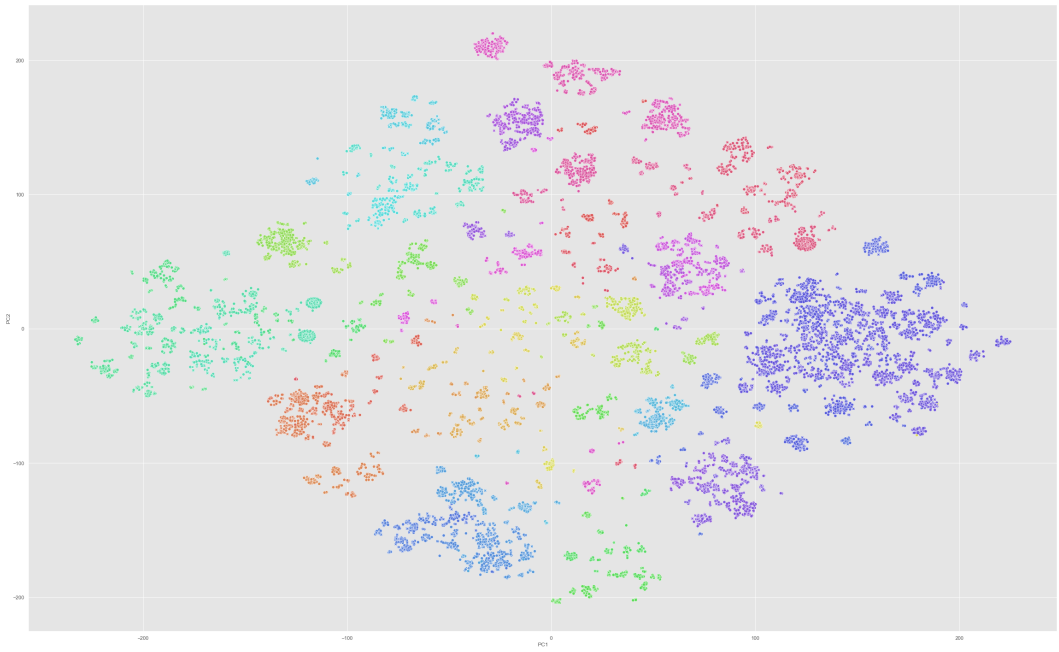


Figure 14. T-SNE visualisation based on cosine similarity. Hierarchical clustering using "distance" method before and after noise cleaning

Appendix 5 – Clusters statistics

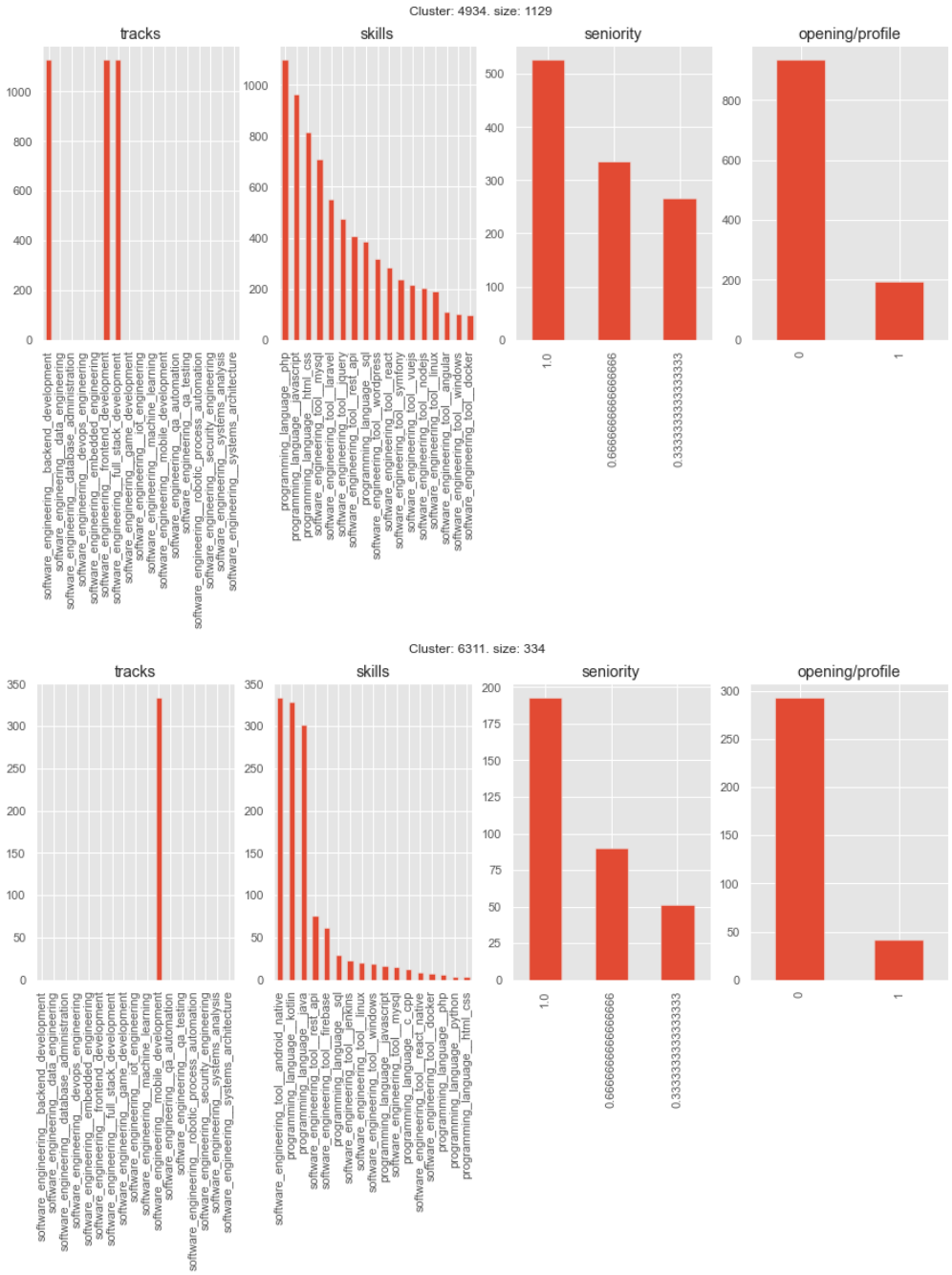
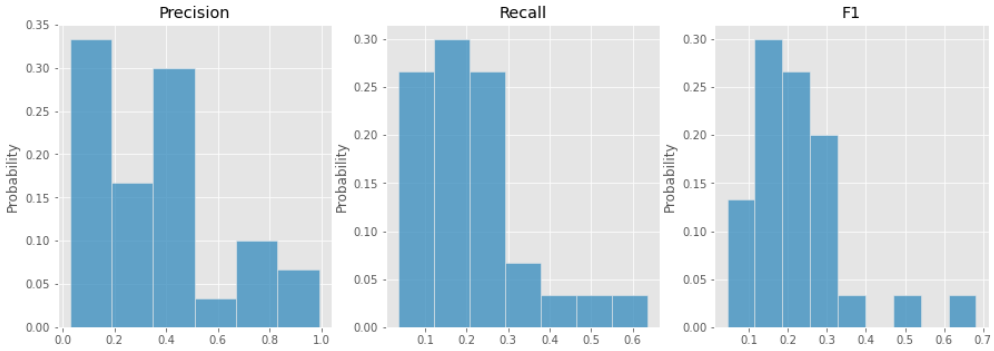


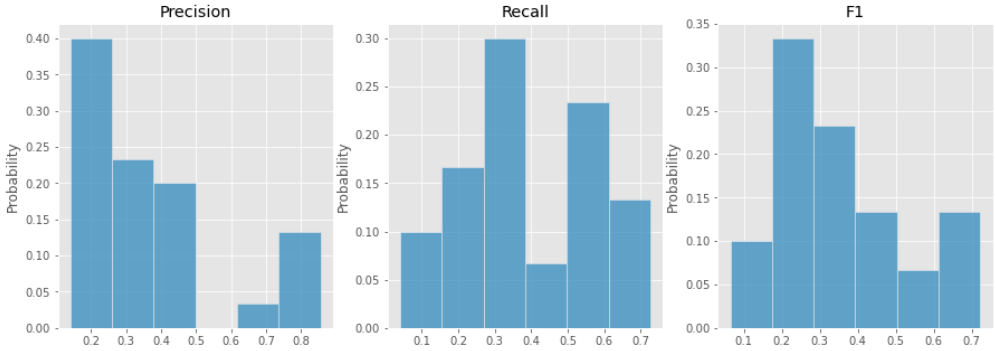
Figure 15. Statistics of two randomly selected clusters

Appendix 6 – Models performance comparison

Word2Vec model:



Naive Bayes model:



KNN model:

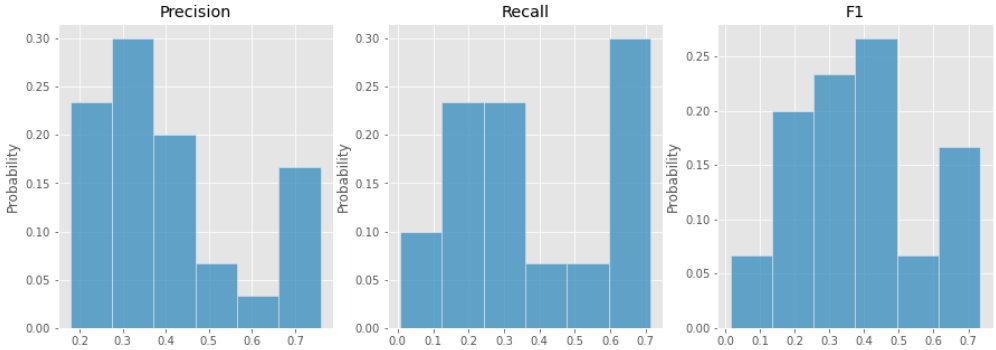


Figure 16. Distribution of performance metrics per skill of different models

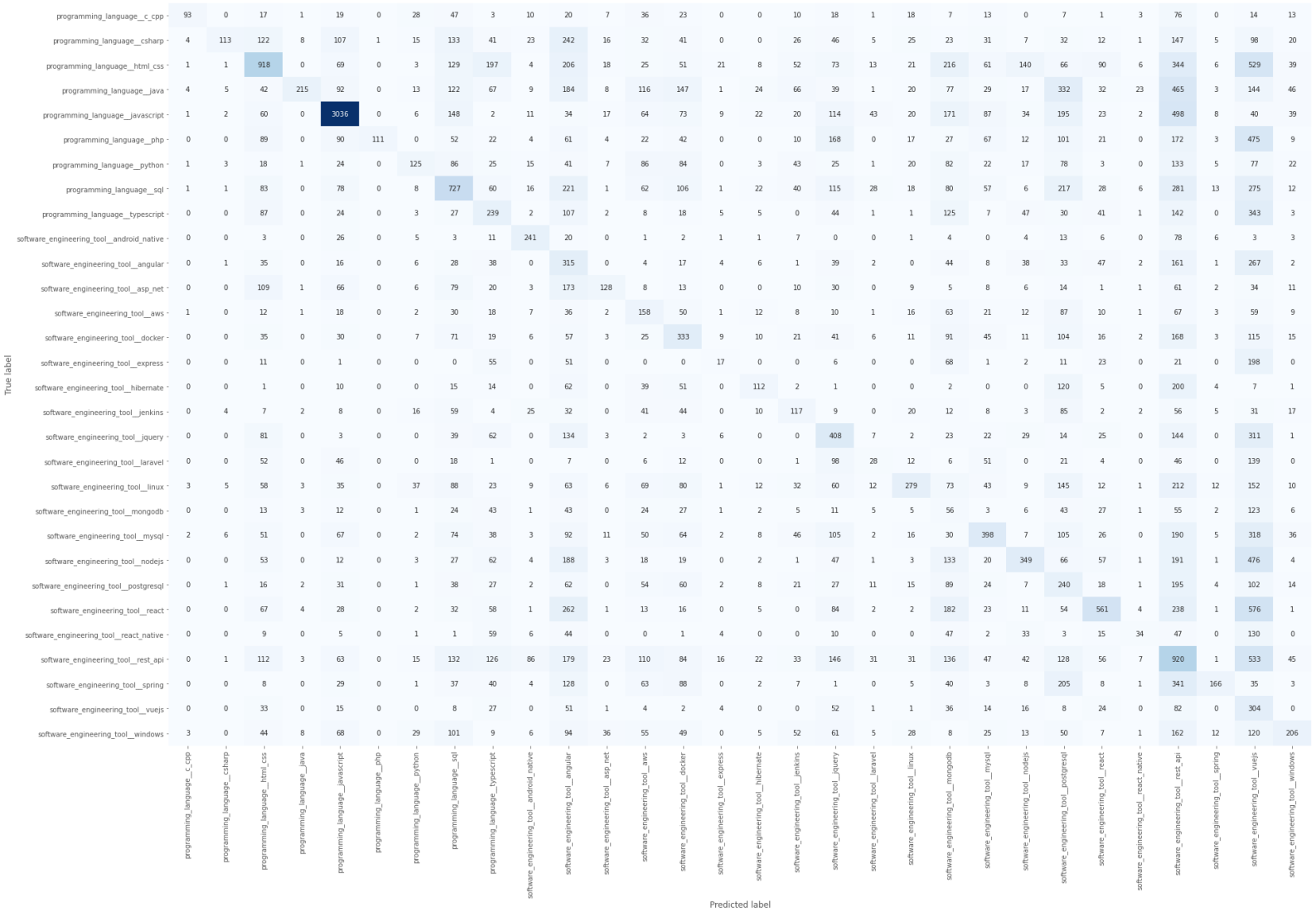


Figure 17. Confusion matrix of Word2Vec based model

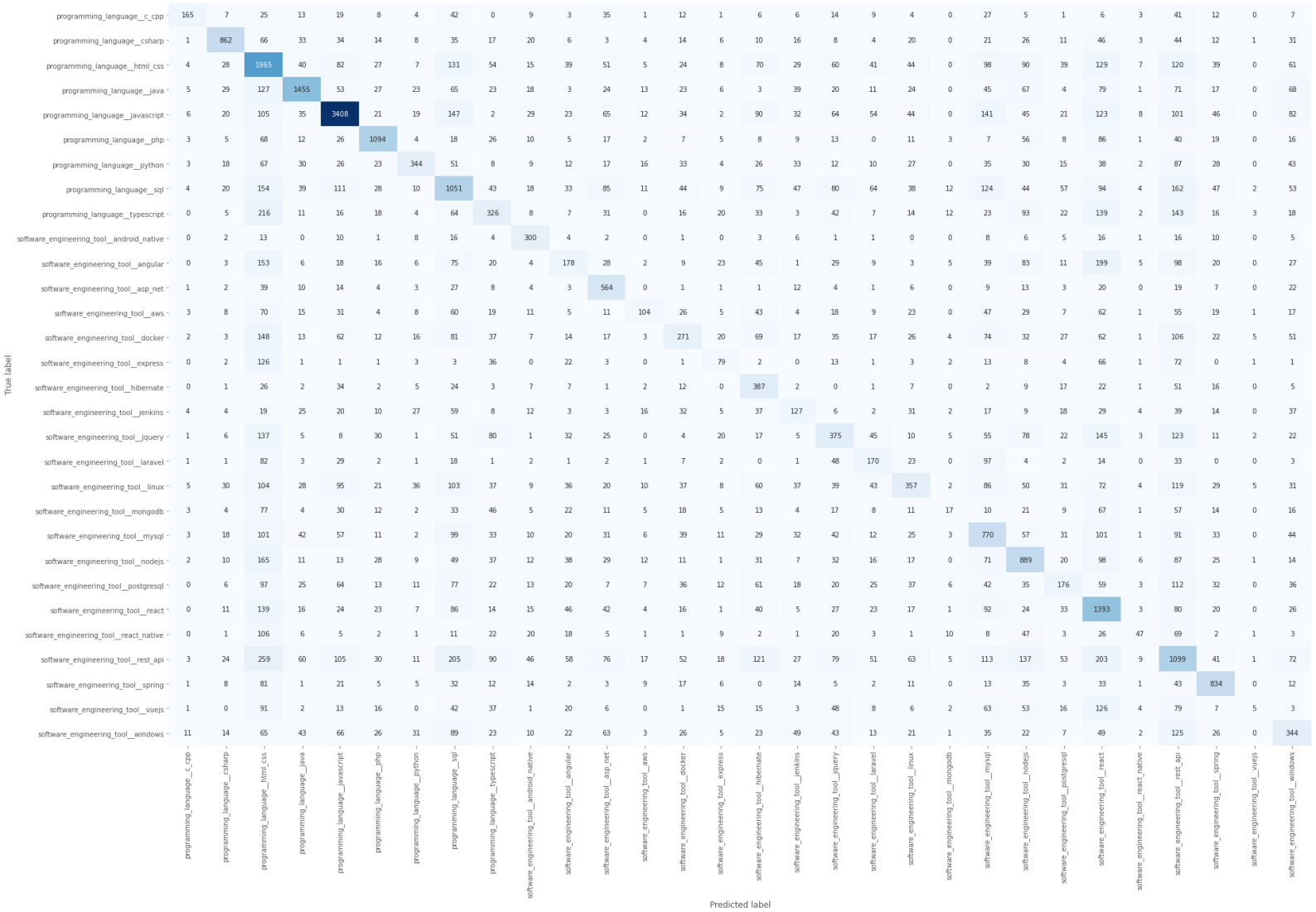


Figure 18. Confusion matrix of KNN based model

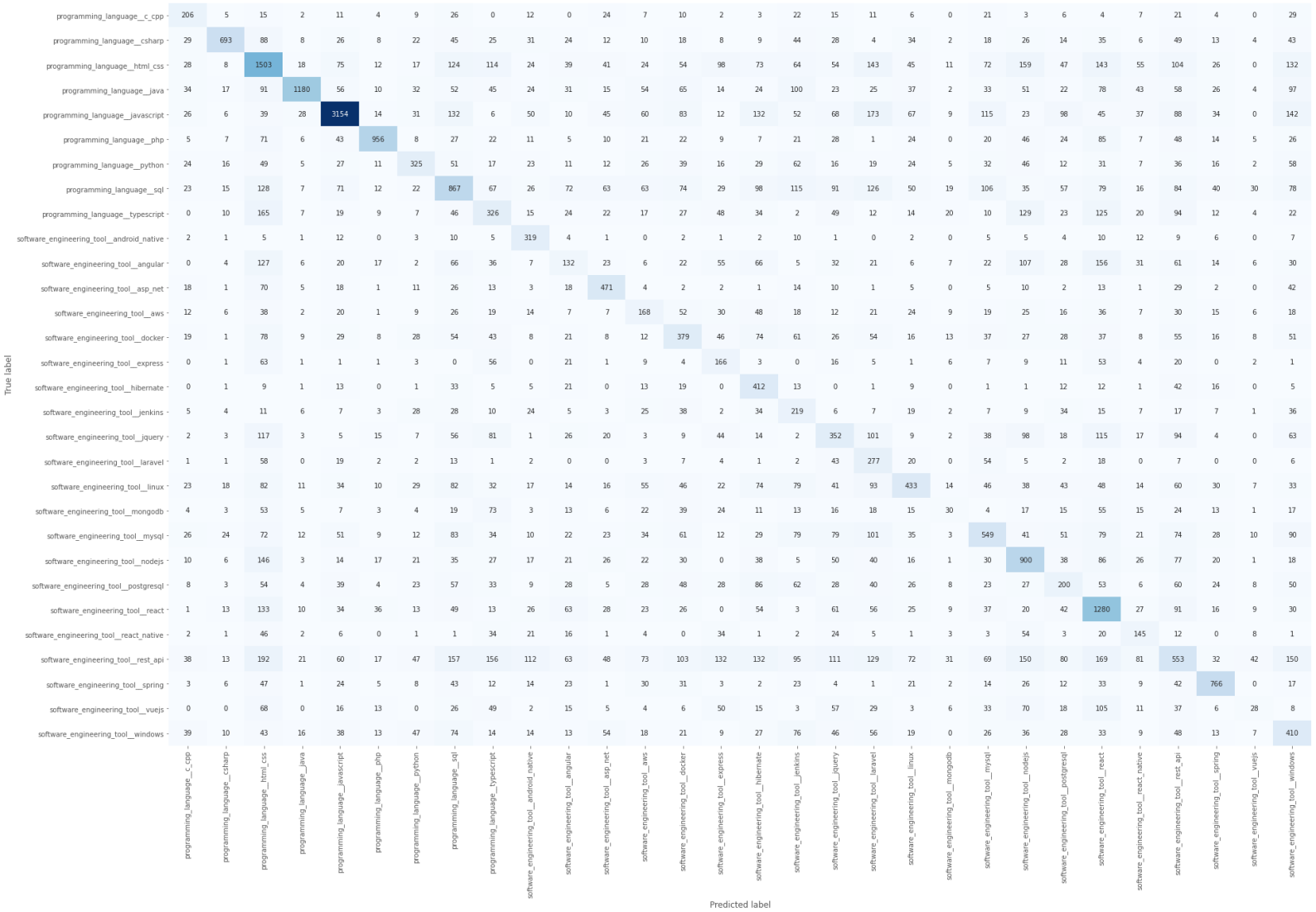


Figure 19. Confusion matrix of Naive Bayes based model