TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Science

Raigo Aljand

Assessing Article Quality in Wikipedia Using Machine Learning Algorithms

bachelor's thesis

Advisor: Jaagup Irve
Chair of Network Software

Tallinn 2014

Abstract

The Estonian Wikipedia has a lot of articles that are of high-quality, but are hard to find from the huge set of articles. The aim of this thesis is to filter out the high-quality articles from the low-quality ones using a machine learning algorithm called logistic regression.

The main problem was filtering out the high-quality articles from the low-quality articles. An algorithm was written using gradient descent to find the logistic regression weights from a matrix of numerical data. Therefore, the main tasks of this thesis were to find known high-quality articles and known low-quality Wikipedia articles, translate them into numerical data, train the machine learning algorithm in a small enough number of iterations and validate the accuracy of this algorithm.

Research shows that the Estonian Wikipedia has a category for hand-picked high-quality articles and a way to obtain a random article, which will labeled as low-quality. Training the algorithm with those results, the accuracy of the result is enough to filter out high-quality articles out of all of the Estonian Wikipedia.

List of Tables

List of Figures

Contents

Introduction

The Estonian Wikipedia has a lot of low-quality articles because there are not enough editors. The aim of this thesis is to make the lives of the editors easier by sorting the articles to high-quality and low-quality. The editors can then focus more on the low-quality articles and less on the high-quality articles.

Main questions I had to solve before writing the algorithm was:

- What features of the article will weigh in calculating whether an article is high-quality or low-quality?
- What articles to label as high-quality and what articles to label as low-quality in machine learning?
- What is the most reasonable way to accomplish this task?

Accordingly, the work was divided into 5 steps:

1. Researching the tools.
2. Determining the features.
3. Collecting the test data.
4. Implementing the algorithm
5. Validating the result.

In the tools section, the prior work and tools used in creating this algorithm will be described. The theoretical base for each tool and then the tool itself will be described. Even though there are a lot of tools, the only choosable part was the programming language. Python was selected because of prior experience with the language, the current popularity and the availability of prior tools.

In the implementation section, the prerequisites, the interface, the implementation of the algorithm and the infrastructure required and the results will be described.

1 Tools

1.1 Programming language

Programming language is a formal language with a set of rules about how the computer should behave. Programming languages usually have syntax and semantics. The syntax of the language is usually considered the grammar of the language. For example, if there is a syntax problem in the program, the computer doesn't understand the command and stops. The semantics of the language is considered to be the vocabulary and meaning of the language. If there is a semantical problem, then the program is valid and understandable to the computer, but it is not what the programmer wished to happen. The program will react rather unpredictably. Syntactic sugar is a feature that the programmer can easily implement it in the language and is there for the convenience.[20]

Any sufficiently complex program or programming language needs to hold and manipulate data. Because holding and manipulating only bits and bytes is uncomfortable and prone to errors, more abstraction is required. Dividing the data into different types will help with ease of use and early detection of errors. A type is the upper bound of a range of values that a variable can assume. In a typed language a variable can be given a nontrivial type while in untyped languages a variable is not limited to a type. Strongly checked languages will give an error in case of mistyping and weakly checked languages will not check for such errors and might produce type-related bugs. Statically checked languages check for type errors during compile time and dynamically checked languages will check during the runtime.[3]

Programming languages are usually divided into two categories: high level languages and low level languages. The difference is that low level languages are designed more around how the computer works, while high level languages are designed more around the productivity of the programmer. Because high level languages need to do more translation between what the programmer wants and what the computer accepts, they are usually slower than low level languages. Another advantage of lower level languages is that they enable the programmer to have better access to specific operations and more understanding how the program is executed. This is very useful when the main goal is optimisation.[22]

Python is a widely-used, general-purpose programming language as shown in figure 1. Python focuses on human readability, simplicity and power to the

programmer.[21, 7] That fact is most likely best expressed by the fact that Python uses indentation to divide its blocks. A language like C uses the curly braces for that and uses the indentation only for clarity of reading. Python, however, forces the programmer to make the program more readable and standardised. Semantically Python is very flexible. It supports functional, object-oriented and procedural programming styles.

Python uses dynamic typing, also called duck typing. Duck typing doesn't check if an object implements some certain interface, but simply tries to call the method or attribute. The principle is "If it looks like a duck and quacks like a duck, it must be a duck".[8, duck-typing] This method of type checking renders interfaces useless.

1.1.1 Imperative programming

Imperative programming is a style of programming with the philosophy of changing the machine's state to the required state, which could be a file in a hard drive or a video on the screen. For that the computer executes a sequence of operations in order, which are specified by the programmer. An operation is some change of state in the computer or calculation of some data.

Procedural programming Procedural programming is a subset of imperative programming. Procedural programming tries to divide the program into variables, data structures and procedures. Procedures are meant to group together abstract operations so they could be reused in different situations. In this case an operation can also be a procedure. Data structures are meant to group together conjoined data so they could be moved and manipulated more easily. Variables are pointers to data. They point to the location of the data in the memory which can be then easily retrieved.

Jan 2014	Jan 2013	Change	Programming Language	Ratings	Change
1	1		C	17.871%	+0.02%
2	2		Java	16.499%	-0.92%
3	3		Objective-C	11.098%	+0.82%
4	4		C++	7.548%	-1.59%
5	5		C#	5.855%	-0.34%
6	6		PHP	4.627%	-0.92%
7	7		(Visual) Basic	2.989%	-1.76%
8	8		Python	2.400%	-1.77%
9	10	⌃	JavaScript	1.569%	-0.41%
10	22	⌃⌃	Transact-SQL	1.559%	+0.98%
11	12	⌃	Visual Basic .NET	1.558%	+0.52%
12	11	⌄	Ruby	1.082%	-0.69%
13	9	⌄⌄	Perl	0.917%	-1.35%
14	14		Pascal	0.780%	-0.15%
15	17	⌃	MATLAB	0.776%	+0.14%
16	45	⌃⌃	F#	0.720%	+0.53%
17	21	⌃⌃	PL/SQL	0.634%	+0.05%
18	35	⌃⌃	D	0.627%	+0.33%
19	13	⌄⌄	Lisp	0.604%	-0.35%
20	15	⌄⌄	Delphi/Object Pascal	0.595%	-0.32%

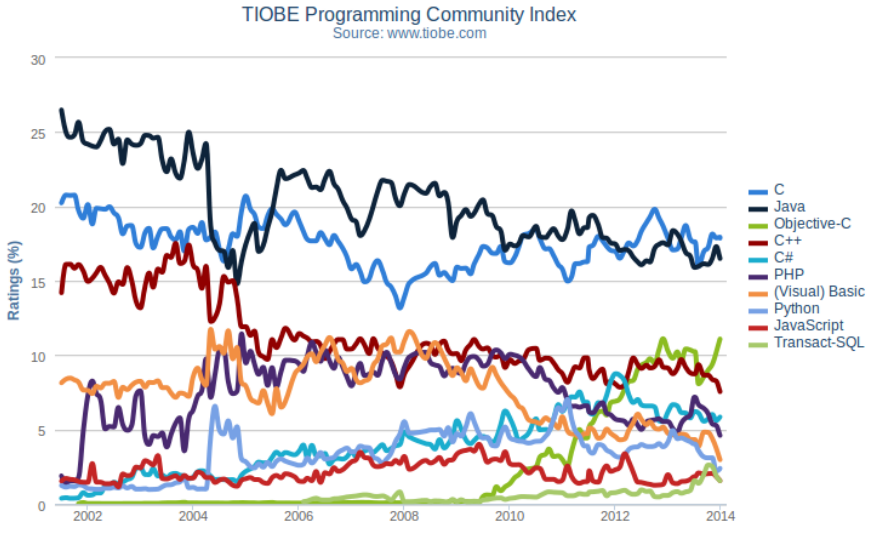

Figure 1: Popularity of Python compared to other languages[2]

Python procedures look like this:

```
>>> def fib(n):    # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print a,
...         a, b = b, a+b
...
>>> # Now call the function we just defined:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

> "The keyword def introduces a function definition. It must be followed by the function name and the parenthesized list of formal parameters. The statements that form the body of the function start at the next line, and must be indented."[6, 4.6. Defining Functions]

Python doesn't make a comparison between functions and procedures.

Each file in python is a module with the same name as the filename, except without the `.py` extension. Each module can import another module or every object in another modules namespace with the `import` command. When the import command is evaluated, the whole file is evaluated once. A directory can also be a module if it contains a `__init__.py` file. In that case the module is initialised with the `__init__.py` file and it's namespace contains the file modules in the directory.

If a variable is nonlocal and the variable in the function is only read, then the interpreter will try to find it from a nonlocal context. However, if the variable is given new value anywhere in the function, the interpreter will assume that the variable is local and the global variable will not be visible. One can make a variable global with the `global` keyword. After that, every reference to the variable will be a reference to the global variable.

Object-oriented programming Object-oriented programming is another subset of imperative programming. Object-oriented programming tries to divide the program into objects that communicate with each other. Each object has fields and methods. Fields and methods are similar to variables and procedures, but they are tied to the object. Fields are considered to be the objects inner state and methods are considered to be the object's

behaviour or object's interface.[19]

The goal of object-oriented programming is encapsulation. Encapsulation means that each object has an inner state, inner behaviour and an interface for other objects to use. An outer object doesn't need to know what is happening within the object. It only needs to know how the object is going to react to an interface procedure. Access protection modifiers are generally employed to better enforce this behaviour. These modifiers are usually tied to a method or field and they describe what other methods are allowed to access these methods or fields. Right to access a field means the right to read or change the field and right to access a method means the right to run the method. An object's methods always have access to its objects fields.[1]

An object can inherit another object. The inheriting object is called subobject and the inherited object is called the superobject. The subobject gets the superobject's fields and methods. A copy of the superobject is created and retained in the subobject. When searching for the subobject's methods or fields and they are not found then the superobject is searched for the field or method. A subobject doesn't automatically have access to the superobject's private fields and methods. The subobject can override the superobject's public methods. The type signature of the method cannot change, but the content or the action of the method is changed.

Class-based programming separates the object into the class and the instance. The class is an abstraction and the classification of the object while the instance is a actual object with actual data. Usually the class holds the behaviour of the object, which includes the constructor. The constructor is a special method, that is called when a new instance is being created from the object. Inheritance works by remembering the inheritance line and then searching for the methods in the right class.

An interface is a class that has no fields and all its methods are public. All of the methods are abstract methods, meaning that the methods have no content or implementation. A class can usually implement multiple interfaces but inherit from only one class. One can't make a instance of an interface, there needs to be an implementing class. An abstract class is a class that has atleast one abstract method. Similar to interfaces, it is impossible to make an instance of them. However they are still classes and and they are inherited, not implemented.

Python has a class-based object-oriented style. Python, however, is more dynamic than a normal static class-based language like Java. After an object has been created from a class, one can still change that concrete object's

variables and methods. Every property is also public. Properties, that the programmer considers private, are usually prefixed with underscores. Each statement is evaluated top to bottom. If there are multiple properties with the same name, then the last evaluated property is remembered.

```
class MyClass:
    """A simple example class"""
    i = 12345
    def f(self):
        return 'hello world'
```

In Python, methods are functions, that get the object's instance in the first parameter, but are called with the first parameter ignored, like this: `my_object.f()`. If there are brackets after the class name and another class's name inside the brackets, the class inherits from the class in the bracket. A method can access the superclass with the `super` function. The super function takes two arguments: the type of the class of whose the super is being searched for and the second is the object.

Multiple inheritance is supported by putting multiple class names in the brackets supported by comas. If called for a property, that a object doesn't have, the environment will try to find the property from the first named superclass recursively until it hits object class and then the next superclass recursively and so on. The object class is the superclass of every class. If there are no brackets or the brackets are empty, then the object class is an implicit superclass.

1.1.2 Functional programming

Functional programming languages are designed around functions. Programming functions are similar to mathematical functions, that it has inputs as parameters and returns a value as output. A function should always return the same output with the same inputs and the inputs should not be changed inside the function. This leads to a particularly stateless form of programming.

Due to the stateless form of this style, functional programming languages usually support immutable datastuctures. Instead of updating a datastructure, it is copied with the new values replaced and the new datastucture is returned. First-class functions and dynamic evaluation of functions are also supported.[18]

Python functions are very similar to Python procedures:

```
>>> def fib2(n): # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series
...     up to n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)    # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)    # call it
>>> f100                # write the result
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

What has changed is that now the results are being collected into a list and then the list is returned with the `return` keyword.[6, 4.6. Defining Functions]

A more functional way of programming would be this:

```
>>> def fib2(n): # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series
       up to n."""
...     a, b = 0, 1
...     while a < n:
...         yield a    # see below
...         a, b = b, a+b
...
>>> list(fib2(100))
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

In this example we don't change the list to collect the results, but instead `yield` the result. When the function is called, instead of returning a final result, it will return a generator. A generator is a simple data structure, that is iterable through only once. The `list` function takes an iterable and returns a list with the iterable's elements.

A `yield` created generator will evaluate the function until the first `yield` keyword, return the result and pause the function. When the next element is asked for, the function is continued and return the result of the next `yield`. When the end of the function is reached, the generator stops.

> "Small anonymous functions can be created with the `lambda`

keyword. This function returns the sum of its two arguments: `lambda a, b: a+b`. Lambda functions can be used wherever function objects are required. They are syntactically restricted to a single expression. Semantically, they are just syntactic sugar for a normal function definition. Like nested function definitions, lambda functions can reference variables from the containing scope:"[6, 4.7.5. Lambda Expressions]

```
>>> def make_incrementor(n):
...     return lambda x: x + n
...
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
```

Python has support for lexical closures, which gives the function a strong reference to the namespace. This excludes the possibility that the namespace will be garbage collected while the function is still in memory. The function has the guarantee that the non-local values will exist even after the enclosing context is deleted or garbage collected.[15]

Since other functional languages have them, Python also has functions `map`, `filter` and `reduce`. `map` applies a function to every item in an iterable and returns a new list with the results of the function. `filter` applies a function to every item in an iterable and returns a new list with the items where the function returned `True`. And `reduce` applies a function for every item in an iterable and returns the value of the last evaluation. In this case the function must have two parameters and the first parameter holds the value from the last evaluation of the function.

A way to avoid using `map` and `filter` is by generator expressions and list comprehensions. List comprehensions are a syntactic sugar to easily create lists. Unlike `map`, list comprehensions don't need to be supplied a function, but can also use arbitrary expressions. A sample list comprehension is: `[2 * item for item in iterable if item % 2 == 0]`. This expression returns a filtered list where each element of iterable is multiplied with 2. The returned list contains only elements that were even before. Therefore the syntax is `[expression for expr1 in sequence1 if condition1 for expr2 in sequence2 if c`
The commands are nested with the right being inside the left and the first expression being the returned innermost expression. Generator expressions are syntactically same, except normal brackets instead of square brackets are

used and they create a generator instead of a list.

Functional programming requires immutable data structures. Fot that python has tuples and named tuples. A tuple is an immutable data structure, that allows packing different data together and afterwards unpack them or access them with an index. Tuples are immutable, meaning they can't be changed.

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

`namedtuple` is a function in the `collections` module that returns a class that inherits from the tuple class. Named tuples have all the properties of a tuple and add on a couple of things:

1. The named tuple has a name and a fitting string representation.
2. Every field in a named tuple is registered with a name instead of an index.
   (a) You can initialise the named tuple with the named tuples.
   (b) You can access the fields of the named tuple by their names.
   (c) You can't construct a named tuple with more or less elements than it expects.

```
>>> Point = namedtuple('Point', ['x', 'y'], verbose=True)
class Point(tuple):
    'Point(x, y)'
```

```
__slots__ = ()

_fields = ('x', 'y')

def __new__(_cls, x, y):
    'Create a new instance of Point(x, y)'
    return _tuple.__new__(_cls, (x, y))

@classmethod
def _make(cls, iterable, new=tuple.__new__, len=len):
    'Make a new Point object from a sequence or iterable'
    result = new(cls, iterable)
    if len(result) != 2:
        raise TypeError('Expected 2 arguments,
                        got %d' % len(result))
    return result

def __repr__(self):
    'Return a nicely formatted representation string'
    return 'Point(x=%r, y=%r)' % self

def _asdict(self):
    'Return a new OrderedDict which maps
    field names to their values'
    return OrderedDict(zip(self._fields, self))

def _replace(_self, **kwds):
    'Return a new Point object
    replacing specified fields with new values'
    result = _self._make(map(kwds.pop, ('x', 'y'), _self))
    if kwds:
        raise ValueError(
        'Got unexpected field names: %r' % kwds.keys())
    return result

def __getnewargs__(self):
    'Return self as a plain tuple.
    Used by copy and pickle.'
    return tuple(self)
```

```
    __dict__ = _property(_asdict)

    def __getstate__(self):
        'Exclude the OrderedDict from pickling'
        pass

    x = _property(_itemgetter(0),
                  doc='Alias for field number 0')

    y = _property(_itemgetter(1),
                  doc='Alias for field number 1')
```

```
>>> p = Point(11, y=22)     # instantiate with
...                         # positional or keyword arguments
>>> p[0] + p[1]             # indexable like
...                         # the plain tuple (11, 22)
33
>>> x, y = p                # unpack like a regular tuple
>>> x, y
(11, 22)
>>> p.x + p.y               # fields also accessible by name
33
>>> p                       # readable __repr__
...                         # with a name=value style
Point(x=11, y=22)
```

1.1.3 Implementation

A programming language by itself is not useful. It also needs an implementation. An implementation is a program that evaluates the syntax of the program into machine actions. There are 3 ways to create an implementation:

Interpretation Interpretation parses the source code and performs the instructions directly

Compilation Compilation is transforming the current source code into a another format that can then be interpreted. Compiled language can also compile into another compiled language which will compile that into another language and so on.

just-in-time (JIT) compilation JIT takes a hybrid approach of interpretation and compilation. While interpreting the program, the JIT compiler will observe what parts of the program are most often interpreted and will compile those parts.

Generally the compiling phase is called the compile time and the interpretation phase is called the runtime. Compiled languages are usually considered fastest, because compilation can heavily optimise the programmer's code, which leads to a faster runtime. Interpretation however is faster and easier to use for the programmer because the compilation step is skipped. For this reason interpreted languages are usually used for scripting. JIT implementations are usually as easy to use as interpreted implementations, but are a lot faster. However because the analysation is fairly complex and runs parallel to the execution of the program, JIT compilers take a lot more memory.[16]

The official implementation of Python is CPython. Every formal change to the language will be almost immediately mirrored in this implementation. It will be supported for a long time will be the most current and up-to-date compared to other implementations. CPython is a bytecode interpreter. It compiles to an intermediate bytecode, which it then interprets. It compiles every time the source file has been changed. This implementation is slow compared to other languages[5], but enables hooking the script to a C module. Since the C programming language is fast,[4] it is possible to write most of the program in Python and write the bottlenecks in C.

The most popular alternative implementation to CPython is PyPy. PyPy is a JIT compiler written in Python. It is popular because of its speed. As of June 7, 2014, PyPy is about 6.2 times faster than pure CPython.[23] There are two disadvantages: PyPy doesn't have hooks into C and PyPy isn't as up-to-date as CPython. As of June 7, 2014, Python 3 support for PyPyis in the beta state while CPython supports Python 3.4.0. Also since PyPy doesn't have hooks into C, it can't speed up the bottlenecks of a program. With simple calculation from [4, 23], C gcc is still about 3 times faster than PyPy.

There are also Jython and IronPython. Jython compiles down to Java bytecode. It allows the programmer to use libraries from Java in his project. IronPython is the same principle, only it compiles to the .NET bytecode. That gives IronPython projects access to .NET libraries.

1.2 Client-server architecture

Client is process that requires some service. Server offers that service. The client and server communicate with HTTP. HTTP is a protocol in which the client sends the server a request and the server processes the request and sends a response. HTTP is purely plaintext. The HTTP request is divided into headers and content. The request has an URL in the header. URL stands for Uniform Resource Locator. It is a way for the client to request a certain page or other resource on the server. Another thing that the URL can contain are extra parameters. From those parameters the server can return the resource in a different form. Usually the request content is empty.

Extensible Markup Language (XML) is a protocol to describe data. It tries to nest data between descriptive tags. The tags can be nested and the tags can have attributes. The tags are not preset, so every user can design it's own way of presenting data. The syntax of a XML tag is `<tag attribute1="value1" attribute2="value2">data</tag>`. A tag can't have the same attribute with different values. data can be normal text or more tags. `<tag />` is shorthand for `<tag></tag>` for when the user doesn't have data to insert.

The HTTP response also splits into a header and content. Inside the content is usually the requested data. A browser is an application that sends HTTP requests to servers and parses and visualises the response to the user. To help with the visualisation and interactiveness of the pages, HTML was created. HTML is an Extensible Markup Language (XML) where the tags are focused on giving text some form of context. For example, the a is a tag for a link and the `h1` is a tag for a level 1 header. The browser also parses the HTML into visual cues and interactions with the user.

HTML, however, lacks the ability change the HTML tags the user is currently seeing. This means, the programmer is unable to interact with the user. For that purpose there exists JavaScript (JS). JavaScript is a dynamic general-purpose programming language. JavaScript works by registering a JavaScript function with an HTML event, so that when that event is fired, that JavaScript function is run. JavaScript interacts with the user by changing the current HTML the user is seeing. Because JavaScript is a general-purpose programming language, unlike HTML, any arbitrary calculation is possible. JavaScript uses Asynchronous JavaScript and XML (AJAX) to send HTTP requests to outside servers. Despite the name, the returned format doesn't have to be XML.

HTML pages are purely intended for browsers to parse and a human to see.

For an another application to get data from a HTML page, it has to web scrape. Web scraping is observing beforehand how a web page is built and later filtering out the necessary data from the HTML. Another way is for the server to offer an Application Programming Interface (API). An API is a way for a program to get data from the server with a HTTP requst in a more formal and machine-friendly form. There are multiple formats for getting the data: JavaSrict Object Notation (JSON), a Domain specific XML and so on.

With JSON the HTTP content is one legal JavaScript Object, which makes it easy to read into JavaScript. Another advantage of it over XML is that there is basic type checking. Javascript allows a value to have a few different types with different notations: a string, a number, another object, an array; and the three constant values `true`, `false`, `null`.[9, 10]

1.3 MediaWiki

A wiki page is a page that owns a title in that wiki and is supposed to aggregate information on the world wide web about the subject of the title. Most of the content in the wiki pages are created or changed by the users of the wiki. MediaWiki is software for a content-classification wiki. A content classification wiki forbids writing personal knowledge and opinions on the wiki page and only allows writing information that can be referenced. MediaWiki has a separarate web page for discussion.[14]

An interwiki link is a link that refers to another wiki page in the same wiki. This way wikis don't have to duplicate information on different pages and just refer to an another page. Links that link to outside the wiki are called external links. Usually external links are in the page as references. There are a special type of wiki pages that are called templates. They are not meant to aggregate information, but that to act as a form template. Whenever a page links to a template, it expands to form a part of the page it is linked on.

MediaWiki has 3 core dependencies. It is completely written in php, therefore it needs php on the system it is running on. It also needs to run on a http server like Apache that is also configured to have php enabled and index.php as a root file. Running MediaWiki is an act of putting the MediaWiki directory in the http server root directory and pointing the browser to the MediaWiki directory. MediaWiki also needs a database service. Officially it supports MySQL 5.0.2+, MariaDB 5.1+, PostgreSQL 8.1+, SQLite

3 and Oracle.[13]

The first time MediaWiki is run, the user is guided through an installation process which will install the database in the database service and create a config file that must be placed in the MediaWiki root directory.

MediaWiki also has an API that exposes the MediaWiki articles to bots without the need to screen scrape. For that in the MediaWiki root directory is the file api.php. Queries made at api.php with the right query parameters will return bot friendly results.[12] For example:

`http://en.wikipedia.org/w/api.php?format=json&action=query&titles=Main%20Page&prop=revisions&rvprop=content`

The format parameter tells what format the bot wants the data to be in. Each format also has a version with a fm added to the end. The fm version pretty-prints, what the bot would have seen, in HTML so it is easy for the programmer to debug with the browser. Different possible formats include JSON, XML, serialized PHP, WDDX and YAML, where xmlfm is the default. However, JSON is the recommended format and all the other formats are deprecated.

The action parameter is the second required parameter. It tells what action the bot wants to do in the wiki. The rest of the query parameters are specific to the action. Most important to us is the query action, which is used to query data from wikipedia. The titles parameter is used to specify pages by name to query. Another way to specify pages is using a generator. Generators generate a list of pages. A bot can give a generator additional arguments by prefixing the parameters with the letter g. For example: `http://en.wikipedia.org/w/api.php?action=query&generator=allpages&gaplimit=3&gapfrom=Ba&prop=links|categories`

This example gets the links and categories of the first three pages. The `generator` parameter tells the API which generator use to generate the pages. The `gaplimit` tells the `allpages` generator how many pages should the generator generate at one time. The `gapfrom` tells what page the API should start listing from. The pages are alphabetically ordered. By default the `allpages` generator uses the main namespace to generate pages from. Another significant generator is `random` that generates pages randomly.

To limit a single user from putting a server on too much of a load. The API allows only a certain number of pages to be queried at once. To have bigger queries, the bot has to use continues. To let the server know, that the bot supports continues, the bot has to add the `continue` parameter with an

empty value to the query. When the query is big enough, the server returns a dictionary under a `continue` parameter. The bot then takes the dictionary and queries again with the same arguments except the dictionary added as query parameters and arguments. The original `continue` argument will be overwritten. If it is not possible to continue, the server will not return a continue parameter.

1.4 PyWikiBot

PyWikiBot is a Python framework which allows an application to communicate with a MediaWiki instance through the MediaWiki API. Firstly the core component of PyWikiBot is the Site object. The site object holds the connection to the MediaWiki instance that PyWikiBot is supposed to query from. It is instantiated without arguments. Instead it takes the instantiation data from the current PyWikiBot configuration.

PyWikiBot is configurable with the `user-config.py` file. On a linux system the `user-config.py` file has to be in the directory `~/.pywikibot`. The `user-config.py` file is created with the script `generate_user_files.py`. It will ask the necessary questions and then generate the `user-config.py` file in the necessary location.

All bots in Wikipedia should have a user account that the bot is using to query and make changes from. All those user accounts should also be flagged as bots, so the admins can make better informed decisions in case of high load. PyWikiBot follows this principle by not being able to run anonymously. When generating user files, PyWikiBot will ask what will be the user name to run with. When running the bot, PyWikiBot will also ask the user for the password. PyWikiBot will remember the password and it is usually not necessary to write the password in again even after a computer restart.

The site that PyWikiBot will query from is dependent on the family and the language. These two are also asked when generating user files. A family is a group of sites that are grouped together according to some theme. A language is the the 2 character code of the language that the site was written in. The url of the site doesn't have to be `ll.family.tld`, where `ll` is the language and `tld` is the top level domain of the site, like how it is with wikipedia. Each site can have it's own url.

1.4.1 Page

The page object holds data about a single wiki page. A page instantation can have 3 arguments. First is the source from which the page will be loaded. The second is the title of the page. The third is the namespace from which the page is loaded. The first argument is obligatory and the instantiating values depend on it. If the source is an another page, then it will make a copy of the page with the title overridden if it is given. If the source is a site, then it reads the title and namespace argument and creates a link from them. If the source is a link instance then it is remembered and the rest of the arguments are ignored.

A link represents a link to a page. A link has 3 instantiation arguments. The first is the text of the link. The second is the site that the link is on. And the third is the namespace to default to if the link text doesn't contain the namespace where the link points to. The text is the only obligatory argument. If no site is given then a new site is initialised. Unless otherwise specified the default namespace will be the main namespace.

Another way to get pages is to use `PageGenerator`s. A `PageGenerator` is a `generator` that queries `Page`s according to a specific criteria. For example, the `RandomPageGenerator` generates random pages from the Wiki. It uses the MediaWiki generator API. There probably is a PyWikiBot `PageGenerator` for each MediaWiki generator.

1.5 Machine learning

Machine learning is used when the programmer doesn't know all about the domain he is writing for. It is then necessary to have the machine learn by itself by some criteria. Machine learning generally divides into two categories: supervised learning and unsupervised learning. Supervised learning is when you have a known data set where for a known input there will be a known output. In unsupervised learning there is no such data set and the programmer is mostly looking for correlation between the inputs. Unsupervised learning is mostly used for data mining.

Logistic regression is a form of supervised learning which is used for classification. Our dataset D will consist of the input of the model, a matrix of the the vectors $\mathbf{x_n}$ and the required outputs y_n for the inputs. A input vector $\mathbf{x_n}$ will consist of the numerical features of the data prefixed with a 1 for the bias. y_n can have only 2 values: 1 or -1. There are more complex forms of

logistic regression that can handle more than two values for y but it is out of the scope of this paper. N will be the size of our dataset.

$$D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_N}, y_N)\}$$
$$\mathbf{x_n} = [\ 1 \quad x_1 \quad \ldots \quad x_d\ ]^T$$

Similar to another supervised learning algorithm linear regression, logistic regression is a linear model."All linear models make use of a "signal" s which is a linear combination of the input vector $\mathbf{x}$ components weighed by the corresponding components in a weight vector $\mathbf{w}$."[**?**]

$$\mathbf{w} = \left[\ w_0 \quad w_1 \quad \ldots \quad w_d\ \right]^T$$

$$s = w_0 + w_1 x_1 + \cdots + w_d x_d = \sum_{i=0}^{d} w_i x_i = \mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x}$$

Linear regression will use the signal directly as output, but logistic regression will pass the signal through a sigmoid or logistic function and treat thats output as the probability that $y = 1$.

$$h(\mathbf{x}) = \theta(s)$$
$$\theta(s) = \frac{e^s}{1 + e^s} = \frac{1}{1 + e^{-s}}$$

As shown in figure 2, the logistic function is good for translating between linear values and probability, because the higher the linear value, the higher the probability of the output value being 1 and the lower the linear value the higher the probability of the output being -1. At input value 0, the probability of it being either value is 0.5.

"We say that the data is generated by a noisy target."[17]

$$P(y|\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = +1 \\ 1 - f(\mathbf{x}) & \text{for } y = -1 \end{cases}$$

We want to learn a hypothesis $h(x)$ that best fits the above target according to some error function.

$$h(\mathbf{x}) = \theta\left(\mathbf{w}^T \mathbf{x}\right) \approx f(\mathbf{x})$$

"It's important to note that the data does not tell you the probability of a label but rather what label the sample has after being generated by the

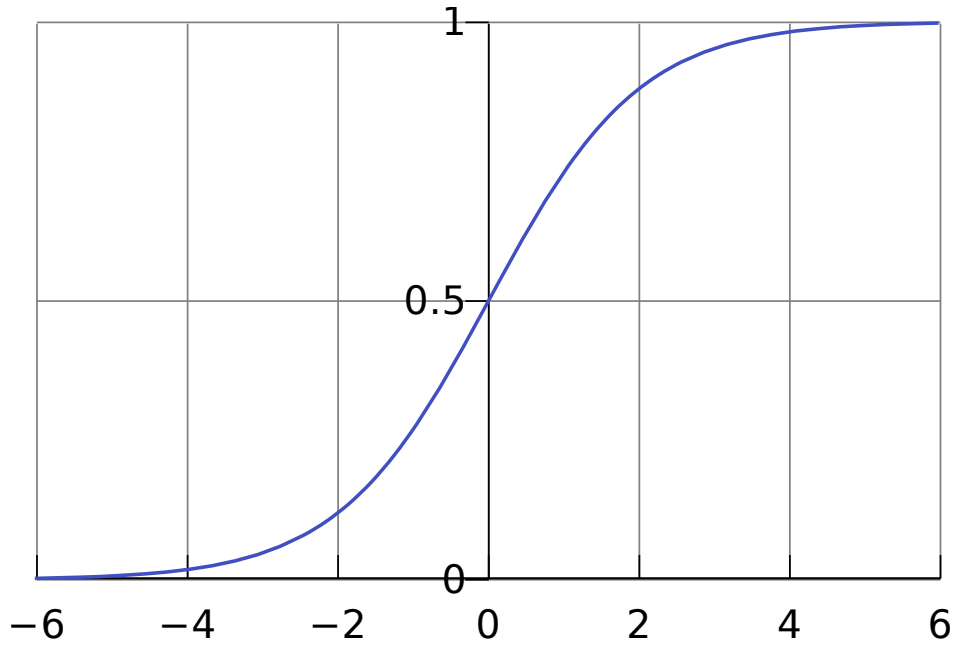

Figure 2: The shape of a logistics curve.

target distribution."[17]. The goal of the training will be to calculate the weight vector w so that it minimizes some kind of in-sample error measure.

$$\mathbf{w}_h = \arg\min_w E_{in}(\mathbf{w})$$

Our error measure will be based on likelihood. Likelihood is the probability of generating the data with a model. Likelihood will be high if the hypothesis is similar to the target distribution. Let's assume that the data was generated by the hypothesis:

$$P(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y = +1 \\ 1 - h(\mathbf{x}) & \text{for } y = -1 \end{cases}$$

$$h(\mathbf{x}) = \theta(\mathbf{w}^T\mathbf{x})$$

Let's try to remove the cases using the property $\theta(-s) = 1 - \theta(s)$.

$$\left.\begin{array}{l} \text{if } y = +1 \text{ then } h(\mathbf{x}) = \theta(\mathbf{w}^T\mathbf{x}) = \theta(y\mathbf{w}^T\mathbf{x}) \\ \text{if } y = -1 \text{ then } 1 - h(\mathbf{x}) = 1 - \theta(\mathbf{w}^T\mathbf{x}) = \theta(-\mathbf{w}^T\mathbf{x}) = \theta(y\mathbf{w}^T\mathbf{x}) \end{array}\right\} P(y|\mathbf{x}) = \theta(y\mathbf{w}^T\mathbf{x})$$

Let's denote an arbitrary hypothesis g, in which case the likelyhood is defined as:

$$L(D|g) = \prod_{n=1}^{N} P(y_n|\mathbf{x}_n) = \prod_{n=1}^{N} \theta(y_n \mathbf{w}_g^T \mathbf{x}_n)$$

To find the best hypothesis, we have to find the best weight vector $\mathbf{w}$.

$$\begin{aligned}
\mathbf{w} &= \arg\max_{\mathbf{w}} L(D|h) = \arg\max_{\mathbf{w}} \prod_{n=1}^{N} \theta\left(y_n \mathbf{w}^T \mathbf{x}_n\right) = \arg\max_{\mathbf{w}} \ln\left(\prod_{n=1}^{N} \theta\left(y_n \mathbf{w}^T \mathbf{x}_n\right)\right) \\
&= \arg\max_{\mathbf{w}} \frac{1}{N} \ln\left(\prod_{n=1}^{N} \theta\left(y_n \mathbf{w}^T \mathbf{x}_n\right)\right) = \arg\min_{\mathbf{w}} \left[-\frac{1}{N} \ln\left(\prod_{n=1}^{N} \theta\left(y_n \mathbf{w}^T \mathbf{x}\right)\right)\right] \\
&= \arg\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} \ln\left(\frac{1}{\theta\left(y_n \mathbf{w}^T \mathbf{x}_n\right)}\right) = \arg\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} \ln\left(1 + e^{-y_n \mathbf{w}^T \mathbf{x}_n}\right)
\end{aligned}$$

We have derived a good form for the error measure, which is the loss function or the average point error.

$$E_{in}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \ln\left(1 + e^{-y_n \mathbf{x}_n \mathbf{w}^T}\right) = \frac{1}{N} \sum_{n=1}^{N} e\left(h\left(\mathbf{x}_n\right), y_n\right)$$

$$e\left(h\left(\mathbf{x}_n\right), y_n\right) = \ln\left(1 + e^{-y_n \mathbf{x}_n \mathbf{w}^T}\right)$$

We minimise the error function using gradient descent. Gradient descent works by moving the current value towards the local minimum. With the derivative, one can calculate the necessary direction and the rough distance of the local minimum from the current value. Therefore training works with the formula:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta \nabla E_{in}(\mathbf{w}_i)$$

Where η is the learning rate. We need the derivative of the point error function and the average point error.

$$\frac{d}{d\mathbf{w}} e\left(h\left(\mathbf{x}_n\right), y_n\right) = \frac{-y_n \mathbf{x}_n e^{-y_n \mathbf{w}^T \mathbf{x}_n}}{1 + e^{-y_n \mathbf{w}^T \mathbf{x}_n}} = -\frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^T \mathbf{x}_n}}$$

$$\begin{aligned}
\nabla E_{in}(\mathbf{w}) &= \frac{d}{d\mathbf{w}} \left[\frac{1}{N} \sum_{n=1}^{N} e\left(h\left(\mathbf{x}_n\right), y_n\right)\right] = \frac{1}{N} \sum_{n=1}^{N} \frac{d}{d\mathbf{w}} e\left(h\left(\mathbf{x}_n\right), y_n\right) \\
&= \frac{1}{N} \sum_{n=1}^{N} \left(-\frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^T \mathbf{x}_n}}\right) = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^T \mathbf{x}_n}}
\end{aligned}$$

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta \left(-\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}_i^T \mathbf{x}_n}}\right) = \mathbf{w}_i + \eta \left(\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}_i^T \mathbf{x}_n}}\right)$$

To lower the number of iterations, each feature of $\mathbf{x}$ should be normalised before it is used for predicting or training the model. Normalising means that the program calculates the standard score of each of the features which is then used instead. The standard score subtracts the mean from the features and divides that with the standard deviation of the values. A values standard score floats around the 0 value and roughly has the same absolute value as other standard scores. When normalising $\mathbf{x}$ for predicting, the mean and standard deviation cannot be enhanced with that data, because then the trained model will not be expecting such data. It would mean comparing two fundamentally different sets of data. It is important to note that the bias variable in $\mathbf{x}$ should not be normalised, because it will end in a divide by zero error.[11]

2 The implementation

The purpose of this project is to separate high-quality articles from low-quality articles.

2.1 Prerequisites

The author used Arch Linux of June 7, 2014 to make this program and hasn't tested installing and running it on other systems. Proceed on your own caution.

This project requires the Python 2 interpreter on the system. As of June 7, 2014 PyWikiBot does not support Python 3 and therefore this paper's code was also not written in Python 3. The Python library numpy is also necessary. One can install them with the terminal command
`sudo pacman -S --needed python2 python2-numpy`.

A user-config.py configuration file must also exist. To generate it, there is a script in the pywikibot folder. It's named `generate_user_files.py` and and it must be run with the Python interpreter. This is a sample installation process:

```
raigo@archofraigo ~/git/wiki-analyse-bot/core (git)-[master] %
python2 generate_user_files.py

Your default user directory is "/home/raigo/.pywikibot"
How to proceed? ([K]eep [c]hange)
Do you want to copy user files from an existing pywikipedia
installation? n
Create user-config.py file? Required for running bots ([y]es,
 [N]o) y
1: anarchopedia
2: battlestarwiki
[...]
26: wikinews
27: wikipedia
28: wikiquote
[...]
33: wiktionary
34: wowwiki
Select family of sites we are working on, just enter the number
```

```
not name (default: wikipedia):
This is the list of known language(s):
ab ace [...] es et eu [...] zh-yue zu
The language code of the site we're working on (default: 'en'):
 et
Username (et wikipedia): AnalyseBot
Which variant of user_config.py:
[S]mall or [E]xtended (with further information)? S
Do you want to add any other projects? (y/N)
'/home/raigo/.pywikibot/user-config.py' written.
Create user-fixes.py file? Optional and for advanced users
 ([y]es, [N]o)
```

Questions without answers use the default answer by pressing Enter.

The `analyse-wiki.py` file needs to have the right to execute. Otherwise one must use the python interpreter to run it. This can be achieved with the command `chmod u+x analyse-wiki.py`

2.2 Interface

This project is run through the Command Line Interface. The user will find the file to run in the core folder. The file is called `analyse-wiki.py` and it needs to be run with the command `./analyse-wiki.py`. The default behaviour of the script is to retrain the machine learning algorithm and filter the good pages from all of Vikipeedia. The command can also take one argument. If the argument is `train`, then the command will only retrain the machine learning algorithm. If the argument is `find`, the command will only search for good pages using the result of the last retraining.

When the program finds a good page, it will print it out on the terminal in a URL format. After the bot has found all of the good pages, it will then write the list of good pages in a file named `good_pages.pkl` using the Python library `pickle`.

2.3 Architecture

Since, PyWikiBot is a framework and not a library, all of the code is in the src folder. In it is another pywikibot folder and the analyse folder. In the pywikibot folder is all of the code for the PyWikiBot framework and in the

analyse folder is the code written for this paper. The interface script is also in the src folder named `wiki-analyse.py`.

Most of the analysis code is programmed in a functional style. There are no classes, only named tuples and most functions don't change the inner state of the parameters, but return a new value. The only imperative part of the program is in the `voidlib.py` file. There each function changes the state of the machine by saving files and outputting messages to the user. Only the highest level functions are there.

The wiki-analyse contains only one `main()` function which will be run when the script is run, but not when it is imported. For that there is the safeguard:

```
if __name__ == "__main__":
    try:
        start = time.clock()
        main(pywikibot.handleArgs())
        pywikibot.output("Run time: " + str(time.clock() - start) + " seconds")
    finally:
        pywikibot.stopme()
```

2.3.1 Machine learning

In the `train_resultlib` module is the TrainResult datastructure, which holds the weights, mean and standard deviation of each feature as 3 lists. The module also has one function `train_result(model_list)`. The funtions takes in a list of `PageModel` and returns the result of training to them. PageModel is a simple datastructure that contains a wiki Page and the label assigned to that page.

The `modellib` module contains the
`predicted_label(train_result, page)` function, that will return the machine's prediction for the Page's label.

The training data consists of the pages in the category "Head artiklid"[1][2] except for the 5 articles that are articles about the category itself and are not examples of good articles. This category is hand built by the Vikipeedia team. An article must fill multiple requirements before it is considered to be good:[24]

[1] `https://et.wikipedia.org/wiki/Kategooria:Head_artiklid`
[2] Good articles in Estonian

Well written It is clearly worded and has the correct spelling. It conforms to the style requirements and doesn't use made-up words or slang.

Factually accurate and verifiable Each paragraph has a citation to used sources and the sources must credible. A good article doesn't contain original research.

Covers the whole subject The main aspects of the subject must be covered while not being derailed to other subjects.

Neutral The subject is presented fairly and without contradiction.

Stabile The article is not often changed because of current arguments or events.

Illustrated with pictures if possible Each picture is marked with copyrights which are not incompatible with Vikipeedia policy. The pictures must be on topic and sufficiently explained. A good article may not have pictures if it is complicated to obtain one.

PageModels with the pages from the "Head artiklid" category are built with the label `GOOD_PAGE`. Then the program asks for the same amount of random pages whose PageModel is initialised with the label `AVERAGE_PAGE`. Most likely the `AVERAGE_PAGE` set will have some very high-quality articles and very low-quality articles besides average articles, but it averages out. Those two sets are then added together and shuffled. Then 70% of it will be used for training and the rest will be used to test the precision of the bot. Sometimes the set the bot is trained with may be biased and the user might want to train the bot again.

Each page has 7 features:

1. Length of the text of the Page.
2. Number of Pages that refers this Page.
3. Number of pages this Page links to.
4. Number of images this Page links to.
5. Number of external links this Page contains.
6. Number of templates this Page links to.
7. Number of categories this Page is in.

Common reasoning says that the higher these features are, the better a Page would be. However we don't know, how one feature weighs against another

feature. That's what the machine learning algorithm figures out. These 7 features with a prefix of the vaule 1 make up the vector x.

To keep the number of iterations small, The values of x are normalised before they are trained or predicted with. It helps keep all the values of x around the same size and around the 0 value. The bias prefix 1 is added after the normalisation so it wouldn't be normalised. This all happens in the below function.

```
def prepare_x(x, mean, std):
    normalised = numpy.divide(numpy.subtract(x, mean), std)
    return numpy.hstack((numpy.ones((normalised.shape[0], 1)),
                         normalised))
```

The result y is the numerical value of the label of the page. If it was a good page, y = 1. If the page was average, then y = -1. The program calculates the probability that y = 1 with the given x vector.

For the function implementing the gradient descent, I am using the example function from [17] with slight modifications:

```
def gradient_descent(z, y, w_h=None, eta=0.5,
                     max_iterations=10000, epsilon=0.001):
    if w_h is None:
        w_h = numpy.array([0.0 for i in range(z.shape[1])])

    # save a history of the weight vectors into an array
    w_h_i = [numpy.copy(w_h)]

    for i in range(max_iterations):
        subset_indices = range(z.shape[0])

        point_error = (- y[subset_indices] /
                       (1.0 +
                        numpy.exp(y[subset_indices] *
                                  w_h.dot(z[subset_indices]
                                          .T)) ))
        grad_E_in = numpy.mean(numpy.tile(point_error,
                                          (z.shape[1], 1)).T *
                               z[subset_indices], axis=0)

        w_h -= eta * grad_E_in
        w_h_i.append(numpy.copy(w_h))
```

```
        if (numpy.linalg.norm(grad_E_in) <=
                    numpy.linalg.norm(w_h) * epsilon):
            break
    else:
        raise Exception("Hit max iterations")

    return numpy.array(w_h_i)
```

2.3.2 Searching

The bot finds the good pages using a brute force mechanism. It requests all the pages from Wikipedia and then tries to predict whether the page is good or average. It skips all pages with exceptions, mostly that would be redirects.

	Training iterations	Tests	Wrong predictions	Error rate (%)	Runtime (seconds)
	766	68	1	1.5	26
	680	68	1	1.5	20
	653	68	3	4.4	15
	645	68	1	1.5	22
	785	68	2	2.9	23
	760	68	1	1.5	22
Average	715	68	1.50	2.2	21.3
Standard Deviation	57	0	0.76	1.1	3.3

Table 1: Training results

3 Results

The most significant result is the error rate of prediction. To calculate it, when the program collects the pages to train the machine learning algorithm, 30% of the pages are randomly selected and separated into a separate set. After the training is complete, the algorithm is then asked to calculate the label of each page in the test set. The calculated label and the known label are then compared and it is possible to tell, whether the calculation was wrong or not. If we add up all the wrong predictions, we get the error rate or the prediction accuracy.

Since the low-quality pages are just random pages, they might have a bias in any feature dimension. However, with big enough sets, the bias will average out. Also, because the test pages are also randomly selected, there might be a bias in them too. Therefore, the error rate can be varied. Table 1, however, shows that the variance is generally small.

Another type of result is the number of iterations of training. The training algorithm gradient descent looks for the local minimum or maximum. It iterates over the result of the last iteration and makes it more precise. It is possible for gradient descent to iterate for a long time or forever, if the required precision or the step between iterations is too high. If the number of iterations reaches 10000, the algorithm is considered to never finish and throws an error. Through trial and error the allowed error rate was fixed at 0.1% and the step size was fixed at 0.5.

The practical use of this algorithm is creating a list of high-quality articles for the Vikipeedia editors.

Summary

The aim of this thesis is to provide a way to filter out the high-quality articles in the Estonian Wikipedia. The best way to do so would be using logistic regression, a machine learning algorithm.

The Estonian Wikipedia has a hand-picked category for high-quality articles. The machine learning algorithm can be trained with these articles. Logistic regression has a weight for each feature or dimension of the article. Logistic regression is also blind to data type and only sees the scalar value. That means that it is impossible to identify whether an article uses a certain kind of template or has certain words or characters in it.

An article has 7 features: characters, pages that refer it, pages that it refers to, images it refers to, external links it contains, templates it links to, categories it is in. These features were selected because they were readily available through the PyWikiBot framework. Because logistic regression accepts only scalar values, the count of each feature was used. A 8th constant weight is added as a prefix to counteract the bias in the data.

An algorithm was written using gradient descent to find the logistic regression weights from the matrix of numerical data. Gradient descent is an algorithm to calculate the local minimum or maximum of a range of data. In this case, we want to find the minimum of the error rate. It iterates over the last result and changes it towards the local minimum. The size of the step of each iteration must be carefully chosen or the algorithm might step over the local minimum without lowering the deviation.

When the data has a great bias or big deviations, gradient descent must go through a lot of iterations to negate that. By normalising the data beforehand, all the features have the same small bias and deviation. This is also done in this thesis, otherwise the iteration count might grow so large, that it is hard to differentiate between a forever looping algorithm and an eventually stopping algorithm.

Validation is done by setting aside 30% of the articles gathered for training and then comparing the label the algorithm predicted for these articles with their true label.

The average error rate of the algorithm detailed in this thesis is 2.22%. This is achieved with averagly 714.8 iterations of gradient descent. This error rate can be considered good enough for the purposes of the editors in Vikipeedia.

One possible improvement is not using the PyWikiBot framework. Because

of the way the PyWikiBot framework's interface works the program queries more times from the MediaWiki server than is required. Each article has a summary page about the article's metadata, which can be screen scraped to get most of the required features with one query. It is also possible to make one specific query through the MediaWiki API to get the required metadata.

Another possible improvement is to subjectively separate bad articles and use that data to teach the machine learning algorithm. In this thesis separating the high-quality articles was done because the high-quality articles were already handpicked by the Vikipeedia editors. The logistic regression algorithm can also be made to accept multiple classifications instead of 2.

I conclude that it is entirely possible to use machine learning to sift out the high-quality articles.

References

[1] Access protection modifiers in java. `http://bmanolov.free.fr/javaprotection.php`. [Online; accessed June 7, 2014].

[2] TIOBE Software BV. TIOBE software: Tiobe index. `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`, March 2014. [Online; accessed June 7, 2014].

[3] Luca Cardelli. Type systems. *ACM Computing Surveys*, 28(1):263–264, March 1996. `http://www.cs.colorado.edu/~bec/courses/csci5535/reading/cardelli-typesystems.pdf`.

[4] Debian. C gcc vs python 3 | computer language benchmarks game. `http://benchmarksgame.alioth.debian.org/u64q/benchmark.php?test=all&lang=gcc&lang2=python3&data=u64q`. [Online; accessed June 7, 2014].

[5] Debian. Python 3 vs java | computer language benchmarks game. `http://benchmarksgame.alioth.debian.org/u64q/python.php`. [Online; accessed June 7, 2014].

[6] Python Software Foundation. 4. more control flow tools — python v2.7.6 documentation. `https://docs.python.org/2/tutorial/controlflow.html`, April 2014. [Online; accessed June 7, 2014].

[7] Python Software Foundation. General python faq — python v2.7.6 documentation. `http://docs.python.org/2/faq/general.`

html#why-was-python-created-in-the-first-place, April 2014. [Online; accessed June 7, 2014].

[8] Python Software Foundation. Glossary — python v2.7.6 documentation. http://docs.python.org/2/glossary.html, April 2014. [Online; accessed June 7, 2014].

[9] Ecma International. Json. http://www.json.org/, October 2013. [Online; accessed June 7, 2014].

[10] Ecma International. Standard ECMA-404 — 1^{st} edition — the JSON data interchange format. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, October 2013. [Online; accessed June 7, 2014].

[11] Richard Lee. Gradient descent and normalized data | richard lee's blog. http://rl337.org/2012/07/03/gradient-descent-and-normalized-data/, July 2012. [Online; accessed June 7, 2014].

[12] MediaWiki. Api:main page - mediawiki. http://www.mediawiki.org/wiki/Api.php, April 2014. [Online; accessed June 7, 2014].

[13] MediaWiki. Manual:installation requirements — mediawiki, the free wiki engine. http://www.mediawiki.org/w/index.php?title=Manual:Installation_requirements&oldid=939059, 2014. [Online; accessed June 7, 2014].

[14] MediaWiki. Manual:what is mediawiki? — mediawiki, the free wiki engine. http://www.mediawiki.org/w/index.php?title=Manual:What_is_MediaWiki%3F&oldid=937775, 2014. [Online; accessed June 7, 2014].

[15] Alex Munroe. Gotcha: Python, scoping, and closures - fuzzy notepad. http://me.veekun.com/blog/2011/04/24/gotcha-python-scoping-closures/, April 2011. [Online; accessed June 7, 2014].

[16] Necrolis. compiler — why does jit'ed code consume so much more memory than either compiled or interpreted code? — stack overflow. http://stackoverflow.com/a/8675550, December 2011. [Online; accessed June 7, 2014].

[17] Viet Nguyen. Fun with logistic regression. http://nbviewer.ipython.org/gist/vietjtnguyen/6655020, September 2013. [Online; accessed June 7, 2014].

[18] Chris Okasaki. Purely functional data structures. http://www.cs.cmu.edu/~rwh/theses/okasaki.pdf, September 1996. [Online; accessed June 7, 2014].

[19] Oracle. Lesson: Object-oriented programming concepts (the java™ tutorials > learning the java language). http://docs.oracle.com/javase/tutorial/java/concepts/. [Online; accessed June 7, 2014].

[20] Terence Parr. What do "syntax" and "semantics" mean and how are they different? http://www.jguru.com/faq/view.jsp?EID=81, May 2012. [Online; accessed June 7, 2014].

[21] Tim Peters. Pep 20 – the zen of python. http://www.python.org/dev/peps/pep-0020/, August 2004. [Online; accessed June 7, 2014].

[22] Lutz Prechelt. Are scripting languages any good? a validation of perl, python, rexx, and tcl against c, c++, and java. http://page.mi.fu-berlin.de/prechelt/Biblio/jccpprt2_advances2003.pdf, August 2002. [Online; accessed June 7, 2014].

[23] PyPy. Pypy's speed center. http://speed.pypy.org/. [Online; accessed June 7, 2014].

[24] Vikipeedia. Vikipeedia:hea artikli nõuded — vikipeedia. //et.wikipedia.org/w/index.php?title=Vikipeedia:Hea_artikli_n%C3%B5uded&oldid=3590326, 2013. [Online; accessed June 7, 2014].