Fussenegger Daniel, BSc

BttFS - Back to the Future Search: Context-Based Ahead-Of-Time Information Retrieval

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Dipl-Ing. Dr.techn. Kern Roman

Institute for Knowledge Technologies
Head: Univ.-Prof. Dipl-Inf. Dr. Lindstaedt Stefanie

Graz, December 2019

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

Abstract

The entry point of this master thesis is the context-based Web-Information-Agent *Back to the Future Search* (BTTFS) which was developed with the goal of shortening the period of vocational adjustment while working on different projects at once as well as providing different functionalities for finding and re-finding relevant sources of information. BTTFS supports the learning of a context-based user profile in two different ways. The first way is to learn the user profile by the use of a cosine-distance function applied on the Term Frequency-Inverse Document Frequency (TF-IDF) document vectors and the second approach is to learn the user profile with a one-class Support Vector Machine (SVM). Furthermore, the Information Retrieval methods Best Matching 25 (BM25), Term Frequency (TF), and TF-IDF, are used on the created model, to determine the most relevant search queries for the user's context. The central question answered in this thesis is stated as follows:

"Is it possible to anticipate a users future information need by exploiting the past browsing behavior regarding a defined context of information need?"

To answer this question the methods above were applied to the AOL-dataset[1], which is a collection of query logs, that consists of roughly 500.000 anonymous user sessions. The evaluation showed that a combination of

[1] AOL, 2017.

the cosine-distance learning function and the TF weighting function yielded promising results ranging between 18.22% - 19.85% matching rate on average, for the first three single word queries that appeared in advancing order on the timeline of the user actions. While the difference in performance between the cosine-distance method and the SVM method appeared to be insignificant, TF and TF-IDF outperformed BM25 in both of the tested scenarios. Regarding to the gained results, it can be stated, that the future information need of a particular user can be derived from prior browsing behavior in many cases, when the context of information need remained in the same context. Therefore, there are scenarios in which systems like BTTFS can aid and accelerate the user's information generation process by providing automated context-based queries.

Contents

List of Figures

List of Tables

1. Introduction

With the introduction of the World Wide Web in 1989, no one could have known, how profound its impact on our lives would be. These days it is nearly impossible to find aspects of daily life that remain untouched by the influences of the internet. The number of people using various types of internet services like social media, E-commerce, education services, information services, and entertainment services is increasing rapidly. While only 1% of the world population was connected to the internet in 1995, the number of worldwide internet users has grown to over 40% (3.8 billion) of the world population in 2018.[1]

Besides the continuous growth of people using the internet also the amount of content and web pages on the internet is advancing. The number of unique hostnames first reached the 1 billion mark in 2014 and had already grown to 1.3 billion in 2018. The tremendous amount of unique hostnames combined with the shift from analog to digital the amount of data that is accessible on the web started to grow tremendously.[2]

In recent years, more new data was created than in the entire previous history of the web and predictions state that there will be a growth of

[1]internetLiveStats, 2018.
[2]internetLiveStats, 2018.

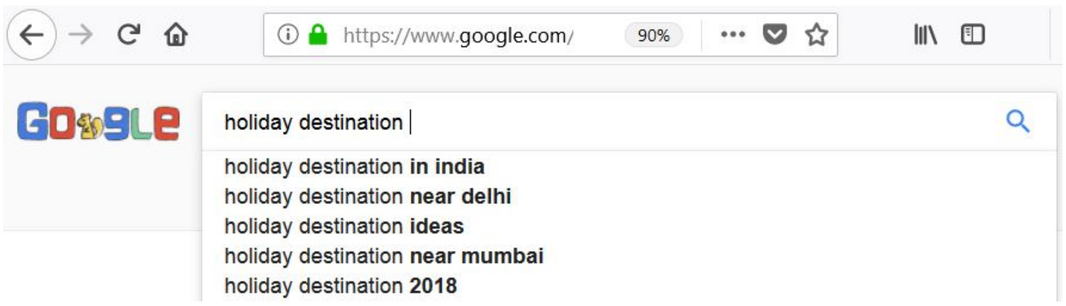
data on the internet to around 40 zettabytes in 2020.[3] Nevertheless, the handling of this large amount of data, also referred as Big Data can be seen as challenging.[4] Due to the quantity and mainly unstructured form of Big Data, traditional software seems unreasonable to handle Big Data properly. Moreover, the methods of Information Retrieval need to adapt, to keep up with the changing environment. Therefore, in the fast pace of the digital age, there is a great need for technologies which assist the people to save time in their everyday life.

Regarding the retrieval of information, popular search engines like Google, Bing, and Yahoo perform very well already. Nevertheless, the search queries that are entered into a search engine are mainly written by humans and are hence inadequate and error-prone in many cases. Although most search engines offer search suggestions or auto-completion which are mainly based on the personal browsing behavior, on search engine query logs, on the location of the user's computer and the most common phrases other persons searched for, there are still cases in which these features do not perform very well.[5][6]

The context in which a search query generation is performed can be stated fundamental for the result. If someone researched on "India" for a project and then changed the personal context by searching for a holiday destination, it is very likely that a search engine will suggest terms regarding holidays in India, while the change of the user's context was not recognized. This behavior can be seen in Figure 1.1, where the Google search engine suggests to complete the phrase "holiday destination" with "in india", "near delhi",

[3]Forbs, 2018.
[4]Anuradha et al., 2015.
[5]Bhatia, Majumdar, and Mitra, 2011.
[6]Venkataraman et al., 2016.

or ”near mumbai” within the first five suggestions. As a result, there is a need for the possibility to define a personal context, rather than be forced into a context-based on previews operations made.


https://www.google.com/ 90%

holiday destination

holiday destination **in india**
holiday destination **near delhi**
holiday destination **ideas**
holiday destination **near mumbai**
holiday destination **2018**

Figure 1.1.: This figure shows that Google search engine suggests to complete the phrase ”holiday destination” with ”in india”, ”near delhi”, or ”near mumbai” within the first five suggestions, while India was only a relevant topic in the context of a previous search.

In this thesis the context-based Web-Information-Agent *BttFS - Back to the Future Search* is presented. BTTFS was developed with the goal of shortening the period of vocational adjustment while working on different projects at once as well as providing various functionality for finding and re-finding relevant sources of information. Furthermore, BTTFS enables the user to create profiles for various contexts as well as connecting keywords to the profiles. The system then creates a context-exclusive browsing history. Besides the possibility of viewing all visited pages, it is feasible to mark pages as important, highlight keywords on websites, crawl the web pages to obtain suggestions for other possibly valuable web pages for the active context as well as export as well as import profiles to share the gathered information with peers. Furthermore, the pages which were considered as important by the user act as relevance feedback to assist the learning of the user’s profile, while the crawled web pages are used as test data for the machine learning algorithms, to expand the collection of documents, which hold relevant information for the user's context. The created document collection is then used to create new search queries, which are submitted to the Google search engine to assist the user's information retrieval process.

It should lie in everyone's interest, to use search engines on the internet in the most efficient ways possible. In a time, where humans' impact on the climate can no longer be denied, it is more important than ever, to use resources appropriately. Only in January of 2018, the internet's infrastructure created emissions over 80 million tons of CO_2 and at the same time, around 95 million megawatt-hours (MWH) of electricity were used to maintain the internet and its services. Millions of businesses around the world, make use of search engines like Google on daily bases. There are roughly 3.5 billion searches on Google per day which leads to 1.2 trillion searches per year, while over 1000 computers process every search query that is committed to Google to obtain a result in under 0.2 seconds.[7] Every single search query that was not inserted into a search engine, because the information need on a topic was already satisfied, aids the saving of resources and due to the massive amount of searches people perform on daily bases, the potential that lies within is tremendous. However, also the amount of time that could be saved in our everyday life by performing more specific and context-based searches should not be neglected. If the average searching time of all Google users could be reduced by one second, the total conversed time per year would be over 38.000 years.

Therefore it can be said that the enhancement of working efficiency would lead to an enormous economization potential regarding working hours, capital and CO_2 emissions.

[7]internetLiveStats, 2018.

Thus the central question to be focused on in this thesis can be stated as follows: *"Is it possible to anticipate a user's future information need by exploiting the past browsing behavior regarding a defined context of information need?"*

2. Background

2.1. Organizing Information

The archiving, organizing, and re-finding of information can be challenging, due to the mainly heterogeneous and unstructured form of information. Back in around 3.000 BC, the Sumerians already stored clay tables with a cuneiform inscription at special places and used early classification methods for re-finding particular clay tables.[1] The development, storing, and retrieving of information was always an important part of humanity. However, inventions like paper, the printing press, and computers elevated the need for storing and re-finding information to a new level.[2] While traditional libraries often archive books sorted alphabetically, by genre, author, keywords, or even by color this task is more complex for documents on the Web, due to the unfiltered and heterogeneous form of the data. Therefore in the field of Information Retrieval various approaches, like the usage of weighting functions were developed, to find the relevant information in a collection of documents.

[1]Kramer, 1963.
[2]Singhal et al., 2001.

2.1.1. Weighting Functions

Weighting functions are commonly used to weight the terms in a document of a collection, to retrieve the most relevant documents that are related to a given query. However, in this thesis, the weighting functions are used for extracting a ranked list of keywords out of a document collection to generate search phrases that could be relevant in the context of the collected corpus of documents. While over the years various types of ranking functions were developed, the most straightforward methods regarding Information Retrieval are namely TF and the Boolean model. While TF counts the raw amount of appearances of a search word or search-phrase within the document collections, the Boolean model states merely if a query was present in a document. However, these methods seem inappropriate in many cases, since there is no real consideration of term weightings. For example, the TF approach does not consider, in how many documents of a collection a word occurs. Therefore, common words like "the" or "is" are inappropriate to distinguish relevant and non-relevant documents.[3]

The more advanced methods TF-IDF and BM25 consider the Inverse Document Frequency (IDF), which is a measure of how much information a query term provides. Considering a document collection of N documents were a query term t_i occurs in n_d of them, then the IDF weight for the query term t_i is calculated:[4][5]

$$idf(t_i) = log(N/n_d),$$

which is only one of many variants for calculating the IDF values. Therefore, with IDF, terms are considered more relevant when appearing in only

[3]Schütze, Manning, and Raghavan, 2007.
[4]Schütze, Manning, and Raghavan, 2007.
[5]Salton and Buckley, 1988.

a small subset of the document collection, which means high-weighted terms are more significant to distinguish between relevant and non-relevant documents.[6] By combining the $idf(t_i)$ weight of a term with the frequency the term occurs in a document $f_{t_i,d}$, the TF-IDF weights of each word in a document $t_{i,d}$ can be calculated:[7]

$$tfidf(t_{i,d}) = f_{t_i,d}\, idf(t_i).$$

While there are various types of BM25 implementation, one of the most popular instantiations for calculating the BM25 weights for a term in a document $st_{i,d}$ looks like this:

$$bm25(t_{i,d}) = \frac{f_{t_i,d}\ (k_1+1)}{f_{t_i,d} + k_1\ (1 - b + b\ \frac{|D|}{avDt})}\ idf(t_i),$$

where $|D|$ is the document length in words, $avDt$ is the average amount of words per document, while k_1 and b are free parameters used for optimization.[8][9][10]

2.1.2. Learning User-profiles with positive Relevance Feedback

Most traditional learning functions use positive and negative examples of training data to learn a model. However, there are scenarios in which a model has to be trained in the absence of negative learning data. Applications that are based on Relevance Feedback tend to lower the user's

[6]Robertson, 2004.
[7]Salton and Buckley, 1988.
[8]Robertson, 2004.
[9]Sanderson, 2010.
[10]WikiBM25, 2018.

experience and usability if positive as well as negative feedback is required. Also for huge datasets like the data on the web, there would be a need for a considerable amount of negative training data for algorithms to create the right classifications. Therefore, to not create an overhead of feedback-tasks for the users, BTTFS only uses positive Relevance Feedback to learn the user-profiles.

Both of the methods for learning a user profile, that were applied in this thesis use training data in the form of TF-IDF weighted vectors, were each vector represents one document of the positive training data. Each TF-IDF weight of one of these vectors represents a separate dimension in a multidimensional vector space. This form of representing documents is called the vector space model.[11]

One straight forward approach of determining new documents as closely related to the documents gained with the user's Relevance Feedback, is to calculate the cosine similarity between them. The cosine similarity of two vectors of the vector space model is the cosine of the enclosed angle of the vectors. To determine the cosine similarity of a document vector $\vec{q}$ and a document vector $\vec{d_j}$ as seen in Figure 2.1 the cosine function is applied as follows:

$$cosSim(\vec{d_j}, \vec{q}) = \frac{\sum_{i=1}^{N} w_{i,j}\, w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2}\, \sqrt{\sum_{i=1}^{N} w_{i,q}^2}},$$

where $w_{i,j}$ are the weights of the document vector $\vec{d_j}$ and $w_{i,q}$ are the weights of the document vector $\vec{q}$.[12][13]

[11]Salton, Wong, and Yang, 1975.
[12]Singhal et al., 2001.
[13]WikiCoSDistance, 2018.

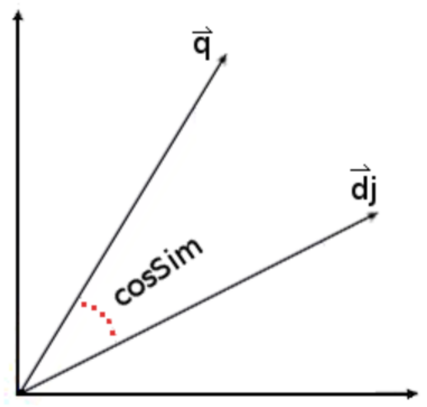


Figure 2.1.: This Figure shows the vectors $\vec{q}$ and $\vec{d_j}$ in the vector space were *cosSim* represents the cosine similarity between them.

Since TF-IDF weights cannot be negative, cosine distance of two vectors in the positive vector space will yield outcomes bound in [0,1], where 0 can be interpreted as not related, while 1 can be interpreted as an exact match. If the cosine measure exceeds a defined threshold, the document will be categorized as closely related to the user's context and is furthermore added to the existing user profile.[14]

Another method for learning a user profile with only positive learning examples is to use a one-class Support Vector Machine (SVM). As an unsupervised learning algorithm one-class SVM is used for learning a decision function. Furthermore, these function is deployed for novelty detection by the classification of new data as ether different or similar to the training dataset. The proportion of outliers and class members plays a crucial role in one-class SVM and, therefore, requires a certain knowledge (or guessing) of

[14]WikiCoSDistance, 2018.

the expected outlier ratio in the interval $(0, 1]$.[15][16][17] Given a set of training data consisting of vectors $x_i \in R^n, i = 1, ..., l$ with no information about the class, the primal problem of one-class SVM can be stated as[18][19] :

$$\begin{aligned} \min_{w,\xi,p} \quad & \frac{1}{2}w^T w - p + \frac{1}{vl}\sum_{i=1}^{l}\xi_i \\ \textit{subject to} \quad & w^T\phi(x_i) \geq p - \xi_i, \\ & \xi_i \geq 0, i = 1, ..., l. \end{aligned}$$

The dual problem is:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha \\ \textit{subject to} \quad & 0 \leq \alpha_i \leq 1/(vl), i = 1, ...l, \\ & e^T\alpha = 1, \end{aligned}$$

where $Q_i j = K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$. The decision function is:

$$sgn(\sum_{l}^{i=1}\alpha_i K(x_i, x - p).$$

[15] SciKitLearn, 2018.
[16] Muller et al., 2001.
[17] Stradling, 2018.
[18] Chang and Lin, 2011.
[19] Dreiseitl et al., 2010.

2.2. State-of-the-art

2.2.1. Machine Learning

Several well-known machine learning techniques like Bayesian classifier, Nearest Neighbor, and Neural Nets to name just a view, can be used to learn a user profile, though most traditional approaches rely on a training set that includes positive as well as negative features. However, for systems which rely on relevance feedback to generate the training data, negative features seem to have an unfavorable impact on the usability, due to the overhead that would be created regarding the user's tasks. Therefore in this thesis, the focus lies on one-class classification methods that support the learning of user profiles with only positive learning examples.

Three different types of one-class classification methods, namely the boundary-based methods, the density estimations, and the reconstructed methods can be defined. However, in all one-class classification methods, two specific components can be found. The one component is a measure for the distance, probability or resemblance of an object to a target class, while the other component is a threshold on this resemblance, probability, or distance. Objects of the test data-set are then considered similar, depended on the used methods, if either the resemblance value is larger than the given threshold, or if the calculated distance to the target class is smaller than the threshold. The main differences between most one-class classification models lie within the definition and optimization of the distance measure or probability measure as well as in the optimization of the used threshold.[20] While several one-class classification methods exist, only a limited set of

[20]Tax, 2001.

various methods are stated in this chapter. However, the chosen methods will cover a wide range of possible approaches of one-class classification.

Boundary-Based Methods

The boundary methods mentioned in this thesis are the k-centers method and the k-nearest neighbor method. In the k-centers method, k small spheres with equal radii are used to cover the dataset.[21] The centers μ_k of the spheres are positioned on the training data objects, to minimize the maximum distance of all minimum distances between the training objects and the centers. To fit the model to the training data x_i, the following error is minimized:[22]

$$\varepsilon_{k\text{-}center} = \max_i \ (\min_k ||x_i - \mu_k||^2).$$

The one-class nearest neighbor classifier was derived from the well known two-class nearest neighbor classifier, to be able to classify data with only positive learning data. The nearest neighbor algorithm stores all provided examples of the training dataset and uses it as its model. For a given test data example z the distance $d(z, y)$ is calculated to its nearest neighbor$y(y = NN(z))$ so that z belongs to the target class if:

$$\frac{d(z,y)}{d(y, NN(y))} < \delta,$$

where $NN(y)$ stands for the nearest neighbor of y and the default value of δ is 1. For the implementation of the one-class k-nearest neighbor method, the average distance of the k nearest neighbors is considered.[23]

[21] Ypma and Duin, 1998.
[22] Tax, 2001.
[23] Yousef, Najami, and Khalifav, 2010.

Density Estimation Methods

A straightforward approach to obtain a one-class classifier is to create a density model of the data[24] and to use a threshold on this density. A model that applies this strategy is, for example, the Gaussian model. To fit the data well the density estimation methods often need a high amount of test data, owing to the high dimensionality of the samples and the complexity of most density models.[25]

The most straight forward density estimation model in one-class classification is the one-class Gaussian model. It is assumed that the target data form a multivariate normal distribution. Therefore, the probability density function for a test sample z in the multidimensional space, can be calculated as:

$$p(z) = \frac{1}{(2\Pi)^{n/2}|\Sigma|^{1/2}} e^{(-1/2(z-\mu)^T\Sigma^{-1}(z-\mu))},$$

where Σ is the covariance matrix and μ the mean of the target class, which were estimated from the training data.[26]

Reconstructed Methods

The reconstructed method to be mentioned in this thesis is k-means clustering[27]. This method was not constructed for one-class classification in the first place but with the aim to model the data. After a model is chosen and fitted on the data, new objects can be described by a state of the generating

[24] Tarassenko et al., 1995.
[25] Tax, 2001.
[26] Yousef, Najami, and Khalifav, 2010.
[27] Bishop, 1995.

model. While most reconstructed methods tend to make assumptions regarding their distribution in the subspaces or the clustering characteristics of the data, the methods mainly differ on how they define their subspaces or prototypes, their optimization routine, and their reconstruction error. K-means clustering, as one of the most simple reconstructed methods, assumes that the data is clustered. To characterize these data prototype objects μ_k are used which are mainly represented by the closest prototype vector measured by the Euclidean distance. For the optimization of the placing of the prototypes, k-means clustering minimizes following error:[28]

$$\varepsilon_{k\text{-}m} = \sum_i (\min_k ||x_i - \mu_k||^2).$$

The k-means clustering method and k-center method which was mentioned in section 2.2.1 tend to look quite similar to each other they differ in the error which is minimized. While k-means averages the distances to the prototypes of the objects, the k-centers method tries to optimize the spheres regarding centers and radii to accept all data with a main focus on the worst-case objects. Another difference between k-means and k-centers can be found in the way the spheres are placed. While k-means allows free positioning of the spheres' centers, the k-centers method places the centers of the spheres per definition.[29]

[28]Tax, 2001.
[29]Tax, 2001.

2.2.2. Web-Agents

With the invention of the World Wide Web, the main tasks of Information Retrieval started to shift from local document collections towards the enormous amount of unstructured data that can be found on the internet today. Therefore new technologies had to be developed, to fit the new environment that was created by the internet. In September of 1990, the first pre-web search engine called Archie was launched. The purpose of Archie was to index File Transfer Protocol (FTP) archives, to allow the user to find distinct files. As a result of limited storage space, Archie only provided listings of the files but not the content of each document.[30] The first early web search engine that used an indexer and a crawler, which are the essential features of today's search-engines, was called JumpStation and was released in 1993. While JumpStation did not provide any ranking to the found results, it used headings and document titles for the indexing of web pages by the use of a simple linear search.[31] With the launch of the Altavista web search engine in 1995 the usability for searching the web improved tremendously. Altavista was not only one of the first search engines to support natural language queries, but also to generate search tips to aid the users in their search process.[32]

While initial indexing and search tools for the internet provided some satisfaction to the users, their ability to filter, categorize and interpret the gathered data was insufficient. Therefore Web-Agent technologies were developed to apply different Information Retrieval strategies on the inhomogeneous and mostly unstructured data the web provides and also to

[30]P. Deutsch and A.Marine, 2018.
[31]New, 2018.
[32]WordStream, 2018.

overcome the limitations of early search engines.[33] Over the last decades, an increasing number of web search engines, web directories, web search portals, as well as different Web-Agent technologies were developed. Moreover, new techniques and search methods were evolved as well as adapted from previews search engines to generate the powerful search engines we use today.

WebACE

The Web-Agent WebACE was proposed in 1998 with the aim to explore and categorize documents on World Wide Web. The main functionality of WebACE is the automatic categorization of a document set, combined with methods for creating new search phrases. These queries are then used to search for new related documents, which are filtered to retrieve a document set that is most closely related to the starting set.[34]

As seen in Figure 2.2, WebACE consists of a profile creation module, a clustering module, a query generator, a search mechanism, a filter, and a cluster updater. The profile creation module builds a custom user profile, while a user is browsing content on the internet. Among other things, the user interest on a specific document is determined by the time a user spent on viewing it. Once a decent corpus of essential documents is built, WebACE reduces the documents to document-vectors, which are passed to the clustering module. To create the document clusters WebACE uses Association Rule Hypergraph Partitioning Algorithm[35] and Principal Component Divisive

[33]Han, Boley, et al., 1998.
[34]Han, Boley, et al., 1998.
[35]Han, Karypis, et al., 1997.

Partitioning[36]. WebACE then uses and the intersection of TF and document frequency (DF) to generate new search queries out of the created clusters. Furthermore, the generated queries are passed to the search mechanism which searches for similar documents that could be interesting to the user. The obtained documents are gathered, and WebACE offers different options how the newly obtained documents can be used. They can be clustered to filter out the least relevant ones, used to updated existing clusters by the insertion of the new documents, or to completely re-cluster both of the clusters. The user then has the option to add newly found pages to the profile.[37] For a list of the clustering and query generation methods that are used by WebACE see Table 2.1.

Algorithms used (WebACE)	
Clustering Methods	Association Rule Hypergraph Partitioning Principal Component Divisive Partitioning
Query Generation	TF DF
Additional Methods	–

Table 2.1.: Algorithms used by the Web-Agent WebACE

[36]Boley, 1998.
[37]Han, Boley, et al., 1998.

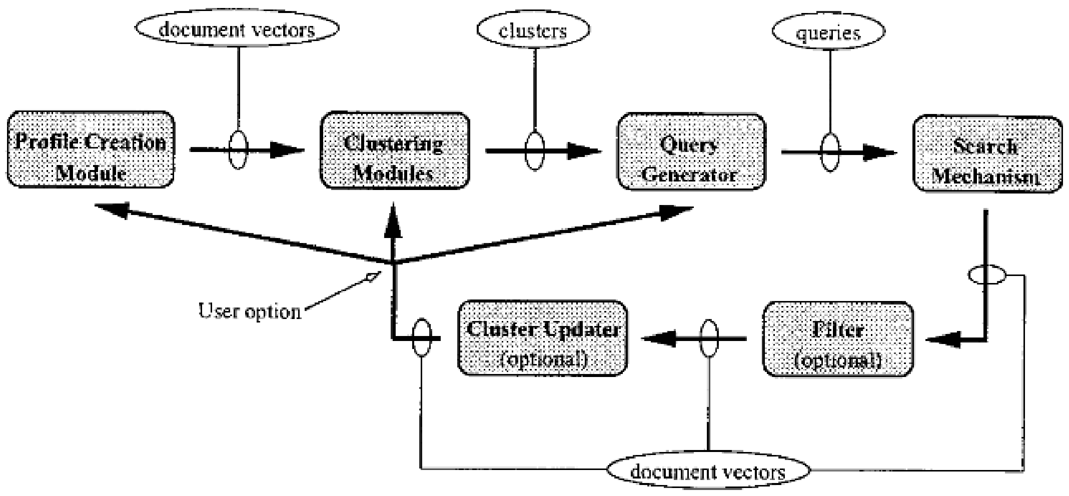


Figure 2.2.: WebACE Architecture

SurfAgent

The Web-Agent SurfAgent was introduced, to generate a profile based information agent to act as a framework for the evaluation of alternative algorithms used for Information Retrieval. As seen in Figure 2.3 SurfAgent follows the basic architecture for personalized web information agents. The main concepts of SurfAgent rely on learning a user profile and applying different query generation methods on the learned document collection. The created queries are then submitted to the Google search engine to verify the resulting pages concerning the matching of the user's interests.[38]

SurfAgent uses term TF-IDF vectors, just like many other Web-Agents, to represent each of the different topics of the user's interest. As the user provides relevance feedback, the system learns new topics of interest and adds TF-IDF vectors of the new relevant documents to the already created vectors. Critical components of SurfAgent are the dissemination thresholds, which are associated with each vector. These thresholds are used when new documents are filtered by the systems. New documents are only considered relevant if the similarity between the vector of a new document and the given profile vectors exceed the associated threshold. To learn a user profile, SurfAgent uses one of two approaches. The first way is to associate relevance feedback with the topic of interest explicitly. To create less overhead for the user, incremental clustering with the Doubling algorithm[39]can be used for the automatic classification of new examples. Regarding query generation, several interchangeable methods like, TF-IDF, TF, and PROBABILISTIC OR were

[38]Gabriel L Somlo and Adele E Howe, 2001.
[39]Charikar et al., 2004.

tested in SurfAgent, to name just a view.[40][41] A detailed list of all essential methods that are used by SurfAgent can be viewed in Table 2.2.

Algorithms used (SurfAgent)	
Clustering Methods	Doubling algorithm Similarity function (dissemination thresholds for TF-IDF vectors)
Query Generation	Uniform (Unif) Boley's method[42] (The intersection of the k top ranked terms of two result sets are selected, where TF is used to on the one set and DF on the other.)[43] TF Probabilistic TF OR Probabilistic OR TFI-DF Probabilistic TF-IDF
Additional Methods	Relevance Feedback

Table 2.2.: Algorithms used by the Web-Agent SurfAgent

[40]Gabriel L Somlo and Adele E Howe, 2001.
[41]Gabriel L. Somlo and Adele E. Howe, 2003.

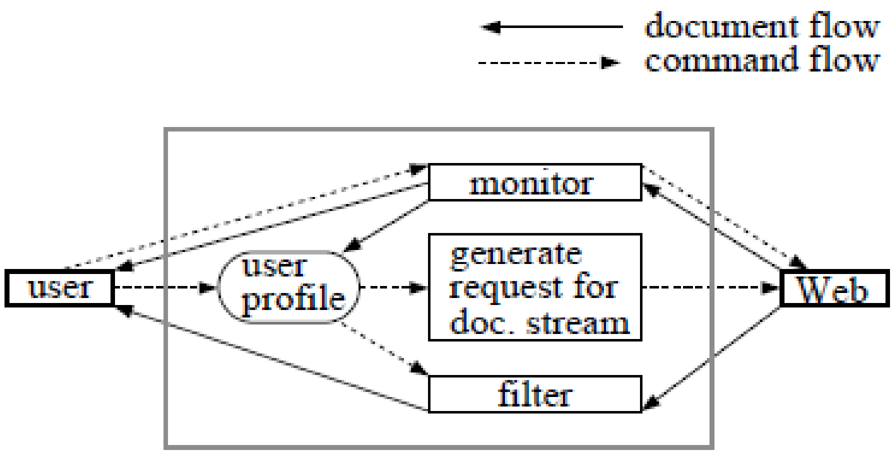


Figure 2.3.: SurfAgent Architecture

WebMate

Hence the fast development of the World Wide Web and the time-consuming task of generating useful information by browsing the Web, ”WebMate: A Personal Agent for Browsing and Searching” was introduced in 1998.[44]

WebMate’s architecture as seen in Figure 2.4 includes a stand-alone proxy for monitoring the user’s actions and an applet controller that handles the interactions with the user. WebMate uses multiple TF-IDF vectors for the different domains of user interests. In contrast to many other systems that use the user profiles for extracting useful information, WebMate uses the generated TF-IDF vectors of the users-interests to learn the user profile incrementally and continuously. To not bother the user with too many unnecessary actions, relevance feedback is only used to identify documents of interest. Therefore TF-IDF vectors with higher weights for particular Hypertext Markup Language (HTML) tags like titles and headlines, as well as a cosine similarity approach, are used. Besides the typical tasks like creating a user profile and catching the most valued information for a user from closely related pages, WebMate also offers the possibility to use the gathered data, to create a personalized newspaper for the user. One way to determine essential pages, which are added to the personal newspaper, is to use the user’s relevance feedback. Subcategories or links from marked pages and their content will be parsed, constructed into TF-IDF vectors and added to the personal newspaper if the cosine distance to the user profile exceeds a certain threshold. The other way is applied if the user does not provide any feedback on essential websites. In this case, WebMate uses the highest-ranked keywords regarding their term-frequency. Given that

[44]Chen and Sycara, 1998.

single keywords are usually too general, a Trigger Pairs Model[45][46] is used to create a more precise search query out of correlating words. Furthermore, the created search query is committed to search engines like Altavista or Yahoo. The pages returned by the search engine are compared to the user profile regarding similarity and are added to the personal newspaper if the similarity is higher then the threshold.[47] A detailed list of all essential methods that are used by WebMate can be viewed at Table 2.3.

Algorithms used (WebMate)	
Clustering Methods	TF-IDF vectors with cosine similarity
Query Generation	Trigger Pairs Model
Additional Methods	Relevance Feedback

Table 2.3.: Algorithms used by the Web-Agent WebMate

[45] Gauch and Futrelle, 1994.
[46] Rosenfeld, 1994.
[47] Chen and Sycara, 1998.

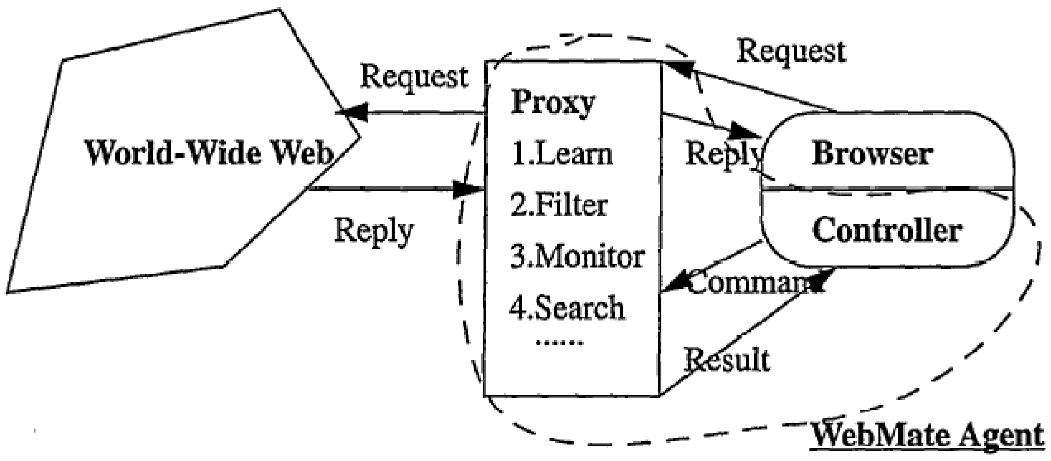


Figure 2.4.: WebMate Architecture

3. BttFS - Back to the Future Search

Back to the Future Search is a context-based Web-Information-Agent. The primary goal of BTTFS is to shorten the period of vocational adjustment on distinct projects. To fulfill this goal, BttFS assists the user by storing information about visited web pages and supports relevance feedback for important pages that were visited. The user profile is continuously learned and extended by the system, with the use of either a cosine-distance approach[1] or a one-class Support Vector Machine. Furthermore, BTTFS performs different ranking techniques namely, TF, TF-IDF, and BM25 on the learned user profile to provide new context-based search queries to the user on demand. Also, BttFS provides the possibility to view a statistics page with high-value information about the profile, regarding keywords, visited web-pages, and suggested web-pages, which could be important for the user's context, while BTTFS's crawling function was used.

The implementation of BTTFS includes a WebExtension for the Mozilla Firefox and uses the WebExtension JavaScript Application Programming

[1]In this thesis, the terms "cosine-distance approach" or "COS" are used interchangeably for the used method of learning the user profile with the help of cosine-distance.

Interfaces (API)[2], which is for the most part compatible with the extension API used by Opera and Google Chrome.[3] A WebExtension can be seen as a software program that allows the customization of a web-browser's content regarding appearance and functionality. The main web technologies such extensions are built on are: JavaScript (JS), Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS).[4] More precisely, extensions consist of so-called content scripts and background scripts. Content scripts can only run in the context of a certain web page and are limited to the standard Document Object Model (DOM) - (API) as well as a very small subset of the WebExtensions APIs. Contrary, the background scripts can access the whole palette of WebExtension JavaScript APIs and hold most of the software's internal logic. For content scripts and background scripts to communicate and pass data to each other a messaging system with event listeners was used. The native messaging API, which is part of the WebExtension APIs, plays an essential role in the implementation of BttFS since it allows the exchange of data between the WebExtension and native program applications. In BTTFS, a python application is used to learn the user's profile, to store the user's data, and to create new search queries that are important for the user's context. Those queries are then passed to the WebExtension, where the new search phrase is inserted into the Google search engine, and the results are displayed to the user.

[2]MDN, 2018.
[3]MDN, 2017.
[4]Google, 2018.

3.1. BttFS - Architecture

BTTFS consists of several components, which can be seen in Figure 3.1. BTTFS is composed of a WebExtension component and a native python application. The WebExtension is used to handle the user interactions with BTTFS, acts as the interface to the web and holds the main logic for manipulating web-content like marking the user's keywords on visited web-pages, creating the statistic page of a profile, and collecting relevance feedback. The native python application component holds the main storage logic of BTTFS and is used for the more complex operations like learning user profiles, expanding learned user profiles, and the creation of search queries out of the user's context by the use of the different ranking functions.

Whenever a user has created a profile, the basic elements of the profile like the username, the corresponding keywords, the specified profile logo, as well as the preferred profile learning method and the preferred ranking function, are stored in the browsers internal storage by the WebExtension.

While the user browses the web, the provided relevance feedback is used to generate the training data for the profile learning algorithms. These training data are passed to the BTTFS's native python application, which stores them. Whenever the user chooses to use the WebExtension's crawl function, the crawled pages are stored if they include any of the context related keywords, which the user associated with the profile. These crawled pages are used to create the test data for the profile learning algorithms and are as well passed to the native application where they are stored.

Furthermore, when the WebExtension is used to generate a search query suggestion for the first time, information about the active user profile like the username, the method to use for the profile learning task, as well as the ranking function type, is passed to the native application. The native application then, based on the transferred information, learns the active user profile with the stored training data and expands the learned profile with the use of the stored test data. The learned user profile is stored by the native application, and with the use of the selected ranking function, a context-based search query is generated. This search query is then passed from the native python program to the WebExtension. BTTFS then opens a new browser window, where the search query is already entered into the Google search engine to display the search query and the gained results to the user. The resulting search query can differ, depending on the used user profile learning algorithm as well as the used ranking method. Whenever a search query was generated, when the profile was already learned, the profile is continuously updated and expanded by the system.

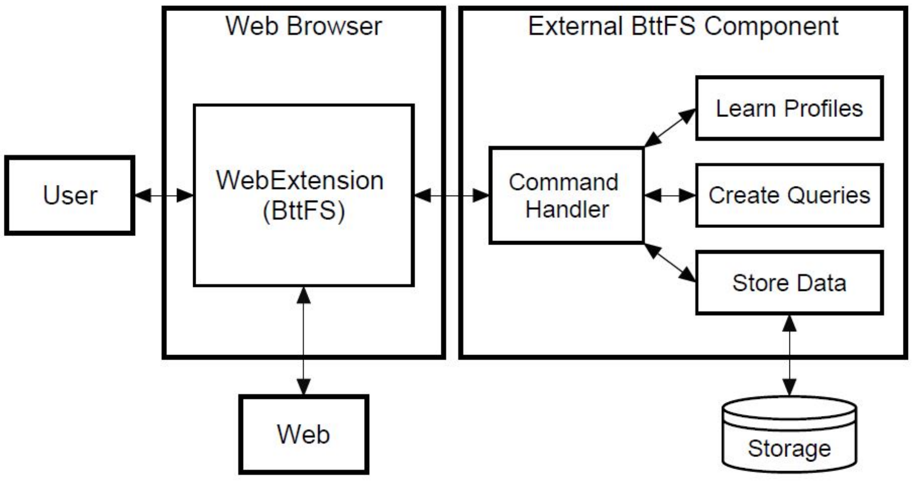


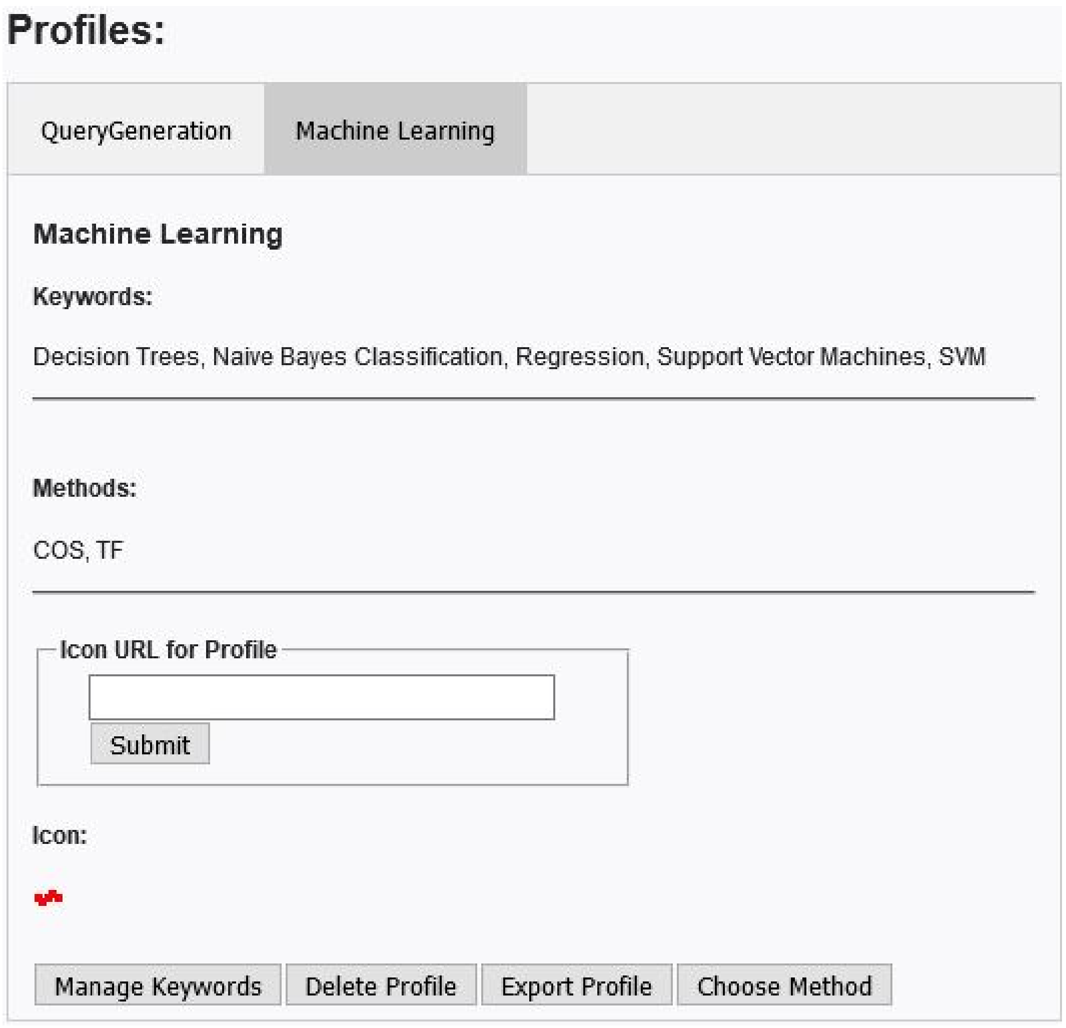
Figure 3.1.: This Figure shows a simplified version of BttFS's architecture.

3.2. BttFS - Functionality

As seen in Figure 3.2, BTTFS allows the user to create a context-based profile and to provide different keywords that should be associated with the profile's context. While it is possible to create variable profiles for different contexts, only one profile can be active at a given time. Furthermore, the user can choose, which algorithm should be used by the BTTFS to learn the user's profile and which ranking function should be applied, to generate new search queries. Also, the user can provide an icon-URL, to customize the profile visually.

Whenever a profile was selected, the browsing behavior of the user is monitored by BTTFS. Not only the visited pages are stored by the system, but also pages the user marked as important. These pages are interpreted as positive relevance feedback and are used to build a collection of learning data, which are used to train the machine learning algorithms, which are used by BTTFS. While the web is browsed by the user, the system offers the opportunity to crawl all links that appear on a distinct web-page. Furthermore, these web-pages are used as the test data for the machine learning algorithms. The user profile is continuously expanded by the time a crawled web-page is classified as relevant by the learning algorithms. Additionally, BttFS uses all crawled pages to create a suggestions table for pages that could potentially be interesting for the users' context based on the TF of the keywords in the visited web-pages. The gathered information regarding the user profile namely the user's browsing behavior, the profile data, as well as the suggested web-pages can be viewed on a profile unique statistics page, as seen in Figure 3.3 .

Moreover, BTTFS supports the import and the export of profiles from other



Profiles:

Machine Learning

Keywords:

Decision Trees, Naive Bayes Classification, Regression, Support Vector Machines, SVM

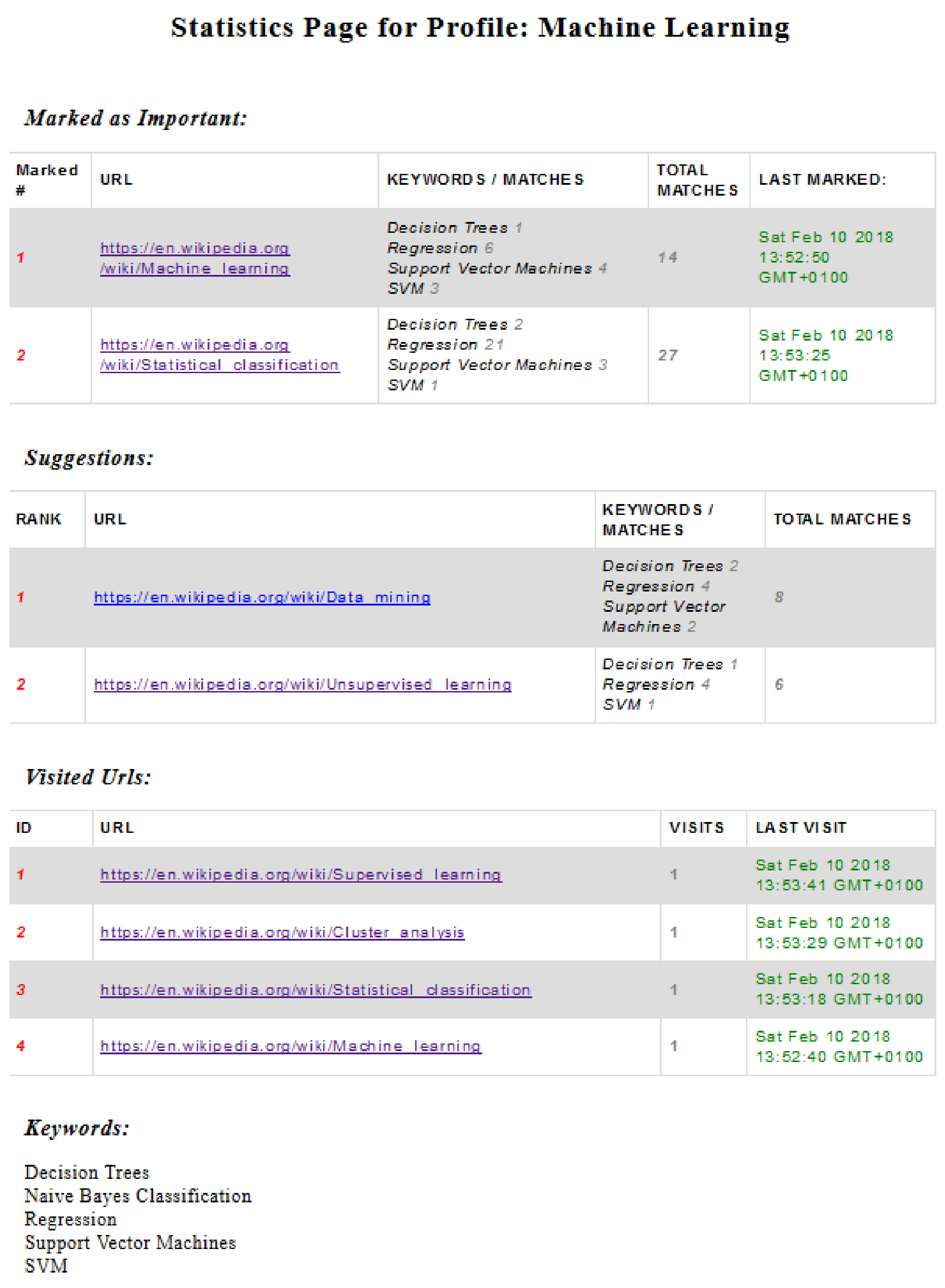
Methods:

COS, TF

Icon URL for Profile

Icon:

Figure 3.2.: This Figure shows an extract of BttFS's settings page. The profiles "QueryGenartion" and "Machine Learning" were created. The active profile is "Machine Learning," and the corresponding keywords are shown. The active method for the learning of the profile is COS, and the and the ranking function for the search query generation is TF.



Statistics Page for Profile: Machine Learning

Marked as Important:

Marked #	URL	KEYWORDS / MATCHES	TOTAL MATCHES	LAST MARKED:
1	https://en.wikipedia.org /wiki/Machine_learning	*Decision Trees* 1 *Regression* 6 *Support Vector Machines* 4 *SVM* 3	14	Sat Feb 10 2018 13:52:50 GMT+0100
2	https://en.wikipedia.org /wiki/Statistical_classification	*Decision Trees* 2 *Regression* 21 *Support Vector Machines* 3 *SVM* 1	27	Sat Feb 10 2018 13:53:25 GMT+0100

Suggestions:

RANK	URL	KEYWORDS / MATCHES	TOTAL MATCHES
1	https://en.wikipedia.org/wiki/Data_mining	*Decision Trees* 2 *Regression* 4 *Support Vector Machines* 2	8
2	https://en.wikipedia.org/wiki/Unsupervised_learning	*Decision Trees* 1 *Regression* 4 *SVM* 1	6

Visited Urls:

ID	URL	VISITS	LAST VISIT
1	https://en.wikipedia.org/wiki/Supervised_learning	1	Sat Feb 10 2018 13:53:41 GMT+0100
2	https://en.wikipedia.org/wiki/Cluster_analysis	1	Sat Feb 10 2018 13:53:29 GMT+0100
3	https://en.wikipedia.org/wiki/Statistical_classification	1	Sat Feb 10 2018 13:53:18 GMT+0100
4	https://en.wikipedia.org/wiki/Machine_learning	1	Sat Feb 10 2018 13:52:40 GMT+0100

Keywords:

Decision Trees
Naive Bayes Classification
Regression
Support Vector Machines
SVM

Figure 3.3.: This Figure shows BttFS's Statistics page for the profile "Machine Learning".

users to shorten the period of vocational adjustment, by benefiting from other user's browsing history. The system also saves the scripts and the HTML of a web page that was visited, marked as important by the user, or suggested by the system. Since the content of web pages can change over time, the storage allows the system to recover the original viewed or suggested web page to regain the already found information. However, the most important feature BTTFS provides to the user is an automated search query generation process that can be started by the user on demand. The system then creates a search query, that could be interesting to the user regarding the context of the user's profile. The automatic query creation is based on the relevance feedback the user provides to the system, by marking pages as important. These pages are furthermore used as the training dataset for the chosen learning algorithm. Additionally, the pages that were crawled by the system are used as test data, which are used to expand the profile's data corpus. The content of the created corpus is then used to generate search queries, which are directly passed to the Google search engine, to provide new relevant web pages to the user.

3.3. BttFS - Creating a Search Query

To generate a custom search query with BttFS for the user's context, as a first step, a user profile has to be learned by the system. After relevance feedback for a vital web-page was given by the user, the HTML of this page is parsed with the python library BeautifulSoup[5]. Furthermore, all scripts of the web-page and other non-text content nodes are deleted, so only the pure text of the web-page remains. BTTFS uses a "window of opportunity" with a length

[5]Soup, 2018.

of 10, which means only words of the web-page are stored, that were either appearing within the range of 10 words in front and within 10 words behind of a keyword that is connected with the user's profile. Therefore, it can be assured that only words are recognized and processed that have a direct connection to the keywords that define the user's context. Additionally, the Natural Language Toolkit (NLTK)[6] and WordNet[7] is used for further processing of the stored words, like the checking of the words for spelling correctness, the stemming of the words, as well as the deleting of stop-words, due to the leak of information they provide. After the processing of all relevant web-pages is finished, the documents are crafted into TF-IDF vectors, where each word of a document represents a dimension in a multidimensional space were the maximum dimensionality is based on the total words in the corpus. Moreover, the TF-IDF vectors are used to learn a user profile either with the COS or SVM method. After a user profile was learned successfully, web-pages that were crawled by BTTFS are used as the test data for the learning algorithms to expand the user profiles data model. The more relevance feedback was given by a user, and the more websites were crawled by the system, the higher the possibility for BttFS becomes for predicting search phrases that are relevant to the user's context correctly. Furthermore, ranking functions like TF, TF-IDF, or BM25 are used on the learned user profile data, to generate a ranked list of the most valuable search phrases, for the user's context. Also, a pseudo-random function is used to generate search queries with a random word length between three and five. The search phrase created tend to be slightly different, depending on the machine learning algorithm and the combined ranking function.

[6]NLTK, 2018.
[7]Miller, 1995.

4. Results

In the course of this thesis, two distinct scenarios were tested on the data of the AOL-dataset[1]. A user session in this dataset consists out of n user actions[2]. For both tested scenarios, the first ten user actions of a user were utilized to learn the profile and to define the context of the user-session. The remaining user actions of the user include the search phrases and keywords that BttFS tries to predict. While the number of user actions of the 796 tested users varies, the methodology for learning a user profile and defining the context of the search remained equal for every user.

The first scenario is called "Exact Query-phrase Matching". The goal of this scenario was to determine if BttFS could predict the exact query-phrase a user would create in the future [3] when only the prior browsing behavior of the user is known, and on the other hand, to identify how well the used learning and weighting functions would perform for this task. If BttFS could predict a user's future search-queries accurately, there would be considerable potential for accelerating and assisting the user's information generation process.

[1] AOL, 2017.

[2] In this context, one "user action" equals one distinct line of the AOL-dataset.

[3] "Future query-phrases" refers to the user actions of the dataset which were not used for learning the user-profile.

The second scenario namely "Single Query-word Matching", was performed to test the accuracy of the used machine learning methods and weighting functions of BttFS for predicting single keywords within the user's future search-phrases, when only the prior browsing behavior of a user is known. Precisely predicting a whole search-query of a user can be very demanding and is getting harder the further a search-query lies in the future of a user-session. However, to assist the satisfaction of the user's information need, the predicting and suggesting of relevant keywords regarding the user's context, can already have a positive impact on the user's information gathering process.

The main algorithms used by BTTFS for learning a user's profile for the two scenarios stated above are the cosine distance approach COS and the one-class SVM approach. For the results which are shown in subsection 4.2 and subsection 4.3, the threshold used in the COS method was chosen as 0.2, which means if the output of the cosine distance algorithm is greater than the defined threshold, a web page was categorized as important for the user's context. Furthermore, the parameters used for the one-class SVM were chosen as the following: nu = 0.1 (lower bound of the fraction of support vectors and upper bound on the fraction of training errors), kernel = RBF (radial basis function) and gamma = 0.1 (kernel coefficient).[4][5]

4.1. The Dataset

The dataset that was analyzed in this thesis is the AOL-dataset. This dataset is a collection of over 500.000 user sessions where each user session is sorted

[4]SCIKit, 2018.
[5]Stradling, 2018.

by the submit time of the query. Furthermore, the dataset provides query log data that are based on real users and consists of around twenty million web queries. Every line of the AOL-dataset includes an anonymous user identifier (ID), the query that was issued by the user, the time at which the query was submitted for a search, the rank of the search result provided by the search engine (only present if a user clicked on the search result), and the URL of the webpage (only present if a user clicked on a search result). An extract of the used dataset can be viewed in Table 4.1.[6] Since the AOL-dataset consists of real data of users, which used a search engine to find content on the internet, the user sessions can vary considerably regarding length and on how many search results were considered relevant by the user. Therefore, to represent a real user that uses BTTFS correctly, only user sessions were inspected, that met a pre-defined set of minimal requirements.

To learn a user profile for every user session of the dataset, the first 10 actions that were performed by a user were examined. Furthermore, the first ten keywords a user entered into the search engine were extracted for each user session. These words were considered essential for the user and got stored by BttFS to define the context of the user's information gathering process. If there was a minimum of four visited webpages within the first 10 actions, these webpages were considered as relevance feedback and were also stored by BTTFS as the pre-set of important webpages for the user's context. Accordingly, BTTFS used these pages as the training dataset to learn the user profile ether with the COS or the SVM method. Subsequently, the first six web-links that each of the stored pages contained were crawled and used as the test dataset to expand the user's profile. For a better understanding

[6]AOL, 2017.

AnonID	Query	QueryTime	IR	ClickURL
1104295	us government	2006-03-09 11:15:05	3	http://www.gpoaccess.gov
1104295	wells fargo	2006-03-09 11:24:07	1	http://www.wellsfargo...
1104295	jerry springer show	2006-03-09 12:16:33	1	http://www.jerryspring...
1104295	solgar	2006-03-09 13:13:54	1	http://www.solgar.com
1104295	national vitamin co.	2006-03-09 20:00:35	2	http://www.nationalvit...
1104295	clipperton island	2006-03-10 07:31:00	1	http://www.cia.gov
1104295	rivilla gigedo is.	2006-03-10 07:32:49		
1104295	revillagigedo is.	2006-03-10 07:33:08	1	http://www.seawatch.org
1104295	revillagigedo is.	2006-03-10 07:33:08	9	http://www.1911encyclo...
1104295	tuamotu is.	2006-03-10 07:41:01	1	http://www.manihi.com

Table 4.1.: Excerpt of the AOL-dataset

of the used parameters for learning a user profile of the AOL-dataset see Table 4.2.

List of Parameters:	Usage:
First 10 actions of a user	Quantity of used data for learning a user profile
First 10 consecutive keywords of **Query**[7]	Interpreted as context-keywords, to define the context of a user session
All webpages (min. 4) of **ClickUrl**[8] within the first 10 actions of a user	Training data for learning a user profile
First 6 webpages linked from each training data sample	Test data for expanding the user profile

Table 4.2.: Parameters for learning a user profile of the AOL-dataset

Furthermore, there are two different ways of how the data got analyzed. For the first scenario that is discussed in Section 4.2, the user actions 11 to 15 were used, to extract the complete queries that the user made per action to check, how many search phrases BTTFS could predict correctly after learning the user profile. For the second way, which is discussed in Section 4.3, also the user actions 11 to 15 were examined, and the first ten unique keywords were stored, to analyze how well BTTFS could predict these keywords after a profile was learned.

It has to be noted that only 796 user sessions of the AOL-dataset provided enough information to learn a user profile with BTTFS. This is since the

[7]The terms "Query" refers to the excerpt of the AOL-dataset in Table 4.1

[8]The terms "ClickUrl" refers to the excerpt of the AOL-dataset in Table 4.1

AOL-dataset contains a large number of user sessions that consist of a small amount of user actions and are therefore not suitable to be analyzed.

4.2. Exact Query-phrase Matching

In this scenario, it was tested how well BTTFS could predict the exact query-phrases of 796 users, which were committed to a search engine after the user profiles were learned. The query-phrases of the users were checked for correctness by the use of a spelling checker and stopwords within the phrases were deleted, due to the lack of information that they provide. Furthermore, only search phrases with a word count between one to five were tested, since these lengths were the most common for the tested 796 users as seen in Figure 4.1.

For this scenario, the user profiles were learned either by the use of the COS method or the SVM method. Further, one of the weighting functions TF, TF-IDF, or BM25 was applied to the data of the user profile to receive the ten best ranked unique keywords for the used method. These ten keywords state the prediction of BttFS for the corresponding user-session. After the predictions of BttFS is created, the future search-phrases of a user were compared with BttFS's prediction. The test system counts the user provided search-phrase as matched by BttFS if all of the words the user's search query consists of are found in the ten keywords that were predicted by BttFS.

The results of this scenario do not only allow to measure how many search-phrases were predicted correctly by BttFS but also to determine which of the applied profile learning technique performed best, as well as which of the combinations of learning algorithm and weighting function outperformed

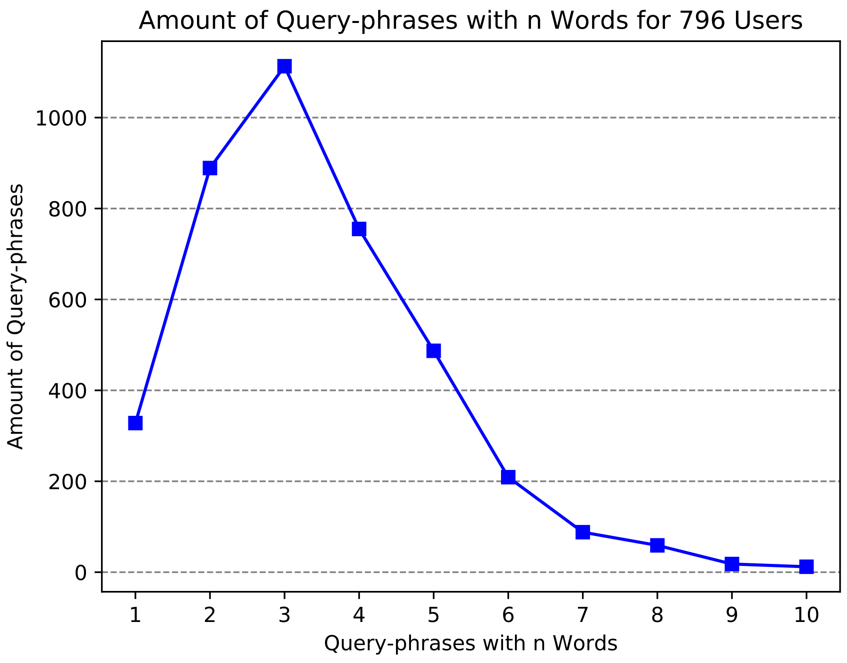


Figure 4.1.: This plot shows the used query-phrases regarding the word count per query-phrase for all tested 796 users.

the others. To figure out, which of the two techniques for learning a user profile in BTTFS yields the better results for this scenario, three different plots where created. As seen in Figure 4.2, if TF is used as a weighting function, the COS method for learning the user profile performed better for search phrases with a word length between one and four, while the SVM method only outperformed COS at a word length of five. However, Figure 4.3 shows, that the SVM method to some extent outperforms COS, when TF-IDF is used for the query-phrase generation. If the BM25 method is used to create the query-phrases, also COS yields the better overall results than SVM as seen in Figure 4.4. Due to these results, it can be stated that for this scenario the COS method performed slightly better than the SVM method for learning the user-profiles when combined with either TF, TF-IDF, or BM25.

To determine which of the utilized query generation methods namely TF, TF-IDF, and BM25 performed best for the search-phrase creation scenario, two plots were generated. These plots allow the comparison of the query generation methods by the relative number of matches that they were able to perform for all 796 tested user sessions.

When the user profiles were learned with the COS method, as seen in Figure 4.5, the TF method for search-query generation even performed better than in the SVM variant. The result shows that TF outperformed TF-IDF and BM25 for each search phrase length by a fair amount. BM25 was again found to underperform, while TF-IDF showed similar results for profiles which were learned either with the COS method or with the SVM method.

As seen in Figure 4.6, if the user profiles were learned with the SVM method, the query generation methods TF and TF-IDF performed equally well for search-phrases with a word count of one and three. TF yielded a slightly better result for a word count of two, and increasingly better results for word counts four and five when compared to TF-IDF. The BM25 method could compete with neither TF nor TF-IDF and seemed to underperform for all tested search-query lengths.

Summarized it can be said, that for this scenario where the exact query-phrase matching for search phrases with a word count between one and five was tested, BTTFS yielded the best results with a combination of the COS method for learning the user profiles and the TF method for creating the search phrases. Other combinations like SVM with TF or TF-IDF also seemed to yield proper results for this task. The BM25 method underperformed for both applied profile learning techniques, which was rather unexpected. For the general performance of BTTFS in this scenario, it can be stated that BTTFS performed best for search-phrases with lower length, while the matching accuracy is reduced with every additional word that extends the length of the search phrase.

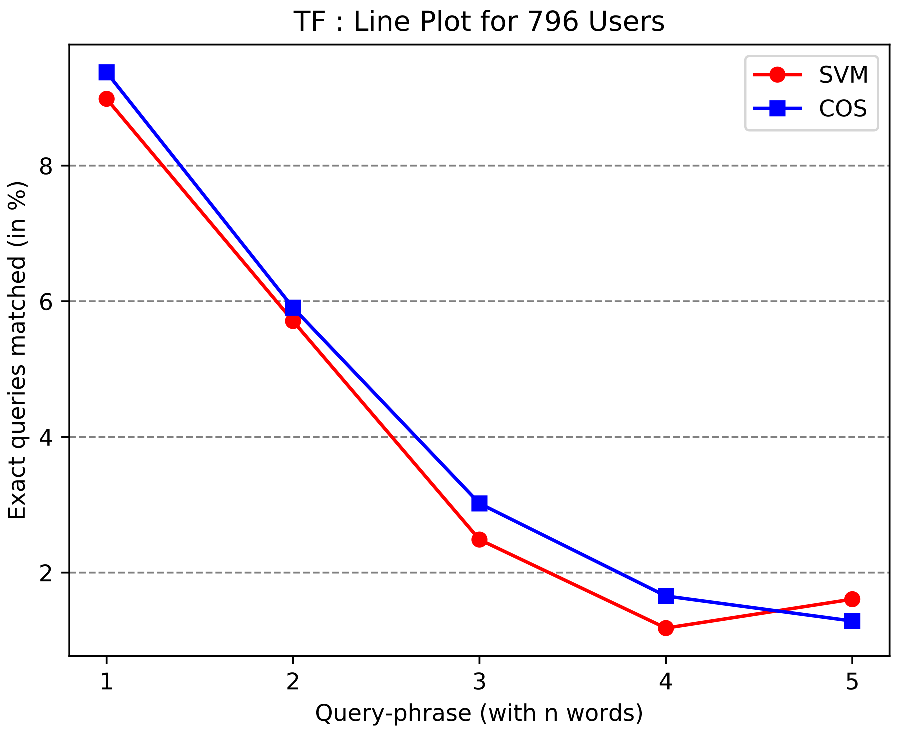


Figure 4.2.: This plot shows how the methods for learning the user profiles, COS and SVM performed relative to each other, while TF was used to generate the predictions. X-axis: Length in words of the exact query-phrases the users submitted to the search engine. Y-axis: Amount of exact query-phrase that was matched by BttFS (in %).

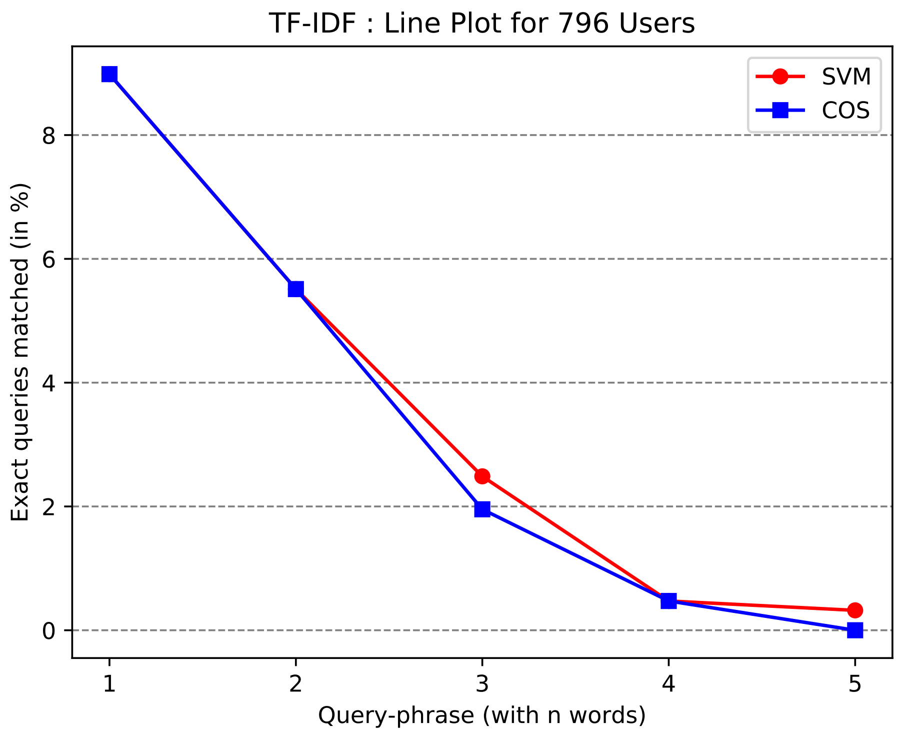


Figure 4.3.: This plot shows how the methods for learning the user profiles, COS and SVM performed relative to each other, while TF-IDF was used to generate the predictions. X-axis: Length in words of the exact query-phrases the users submitted to the search engine. Y-axis: Amount of exact query-phrase that was matched by BttFS (in %).

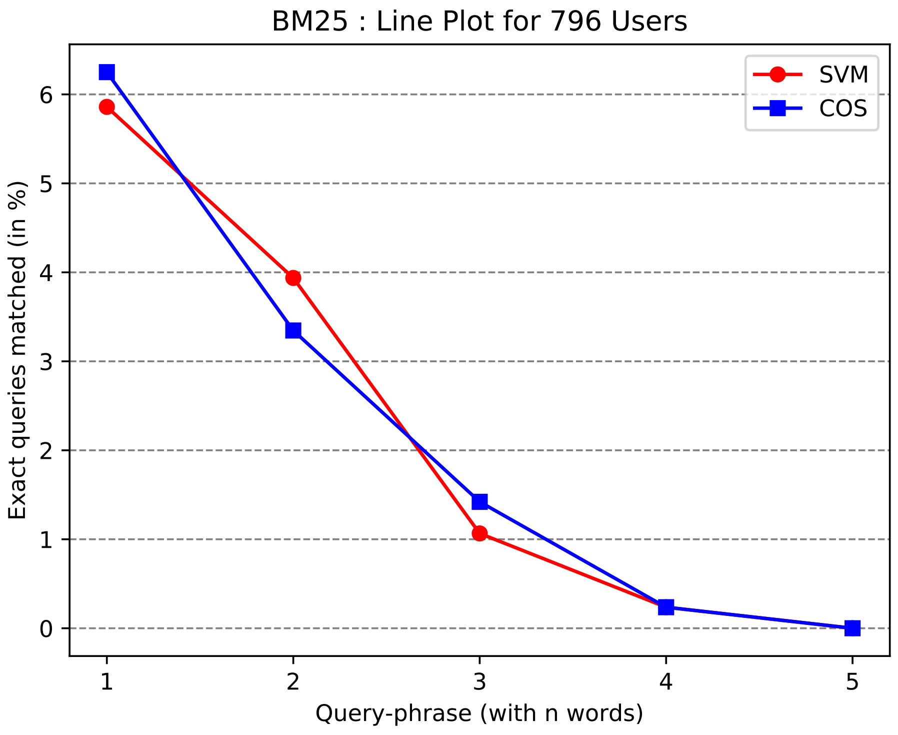


Figure 4.4.: This plot shows how the methods for learning the user profiles, COS and SVM performed relative to each other, while BM25 was used to generate the predictions. X-axis: Length in words of the exact query-phrases the users submitted to the search engine. Y-axis: Amount of exact query-phrase that was matched by BttFS (in %).

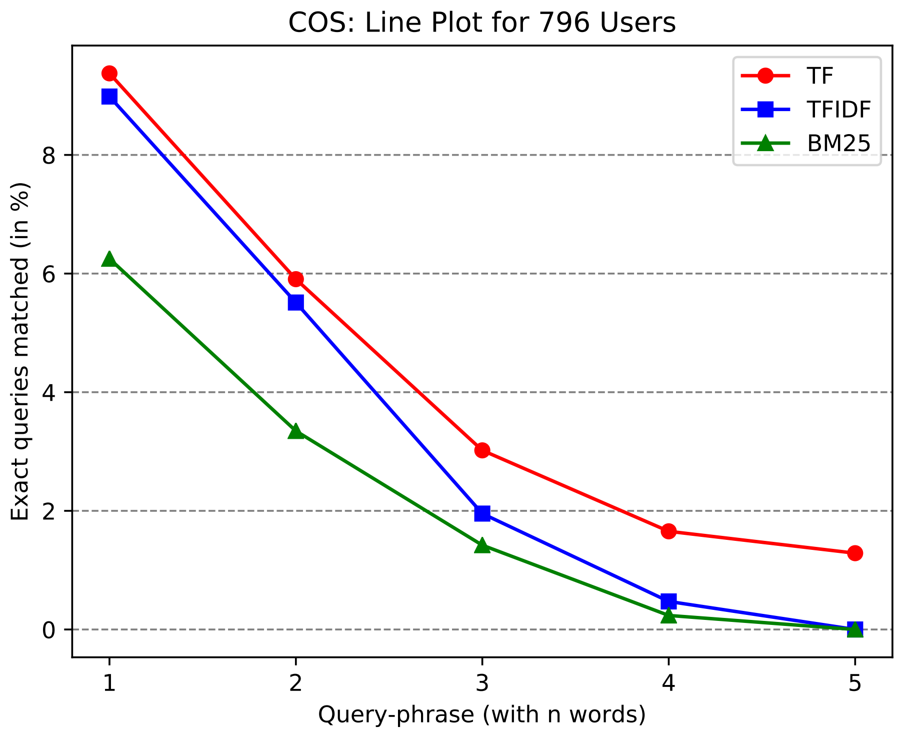


Figure 4.5.: This plot shows the results for the IR methods TF, TF-IDF, and BM25. The user profiles were learned with COS. X-axis: Length in words of the exact query-phrases the users submitted to the search engine. Y-axis: Amount of exact query-phrase that was matched by BttFS (in %).

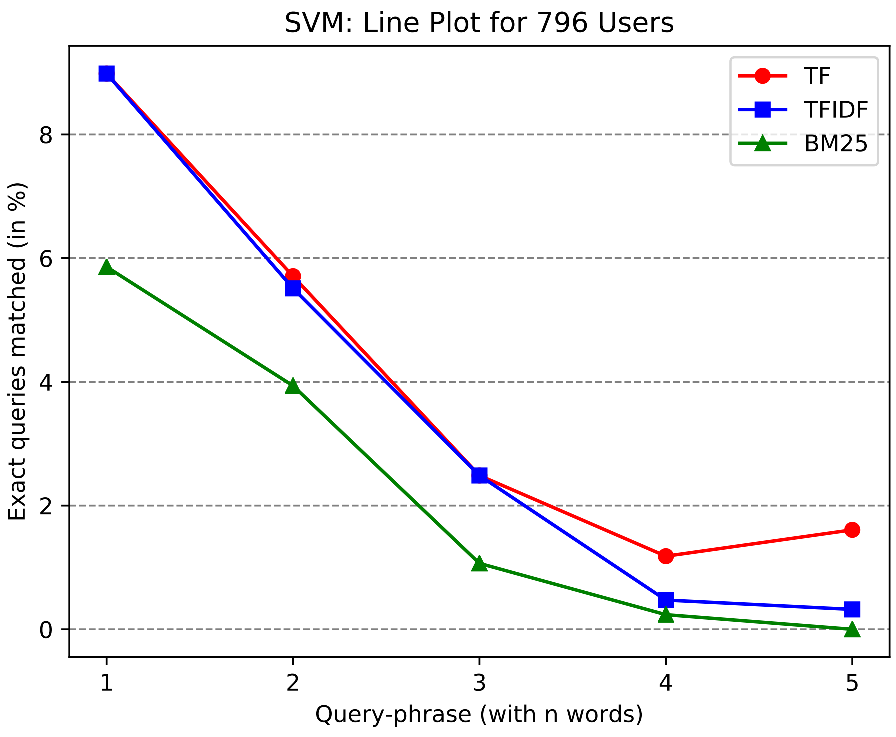


Figure 4.6.: This plot shows the results for the IR methods TF, TF-IDF, and BM25. The user profiles were learned with SVM. X-axis: Length in words of the exact query-phrases the users submitted to the search engine. Y-axis: Amount of exact query-phrase that was matched by BttFS (in %).

4.3. Single Query Word Matching

In this scenario, it was tested how well BTTFS could predict the first ten single query-words that appeared in a user-session for all 796 tested users after the user profile was learned. It was assured that all of the ten mentioned keywords were distinct and checked by a spelling checker. Furthermore, numbers and stopwords were excluded, due to the lack of information they provide, when they do not belong into any context. For this scenario, the user profiles were also learned either by the use of the COS method or the SVM method. Further, the keyword ranking methods TF, TF-IDF, and BM25 were applied to the user profile to generate a list of the ten best-ranked keywords for each of these methods. If at least one of the keywords typed by a user was found within the list of the ten best-ranked words of the keyword generation methods, it was counted as a match for the respective method and the respective keyword position.

The results of the second scenario allow to measure the performance of BTTFS concerning single query-word matching, to determine which of the profile learning functions COS or SVM performed better in this scenario, as well as which of the combinations of learning algorithm and query generation method outperformed the others. To figure out, which of the two techniques for learning a user profile in BTTFS yields the better results for the second scenario, just like in scenario one, three different plots where created. As seen in Figure 4.7, the COS method for learning user profiles yielded slightly better results than the SVM method, when TF was used as the keyword ranking function for the user profiles. While the difference between COS and SVM seems to be marginal for the keywords four to ten, the first three keywords that appeared in the timeline of the user session were reasonably better predicted when the COS method was applied. Contrary to this observation, Figure 4.8 shows, that if TF-IDF was used as the ranking function, COS was outperformed by SVM for nearly every keyword position. Only keywords three, four, five, and ten were equally or better matched by the COS method. As seen in Figure 4.9, for the BM25 ranking function, the SVM method yielded slightly better results than the COS method. As a recap, it can be said that SVM performed better when combined with the ranking functions TF and BM25, while COS outperformed SVM when combined with the TF-IDF function.

To analyze the performance of the utilized keyword ranking functions for the second tested scenario, again two plots were generated. These plots allowed to determine the performance of the keyword ranking functions TF, TF-IDF, and BM25 in a direct comparison. The performance was measured by the relative amount of single keyword matches BTTFS was able to produce for the first ten keywords in the user's timeline, by the use of the different keyword ranking functions, after a profile was learned. This procedure was repeated for each of the 796 tested users. As seen in Figure 4.10, if the user profiles were learned with the SVM method, the ranking function TF outperformed TF-IDF and BM25 for every keyword. While TF-IDF performed similarly to TF for keywords that appeared subsequently in the timeline, TF-IDF was outperformed perceptible for early keywords in the timeline. Like in the first scenario, BM25 performed considerably worse than the other two mentioned ranking functions. This trend also continues for profiles, which were learned with the COS method, as seen in Figure 4.11. The best performance of BTTFS for this scenario can be found in the combination of the COS learning function and the TF ranking function where the first three keywords of the timeline of 796 tested users were correctly predicted between 18% to 20% on average. Furthermore, the results showed, that the further a used keyword lies in the future the less likely it gets for BTTFS to create the right prediction.

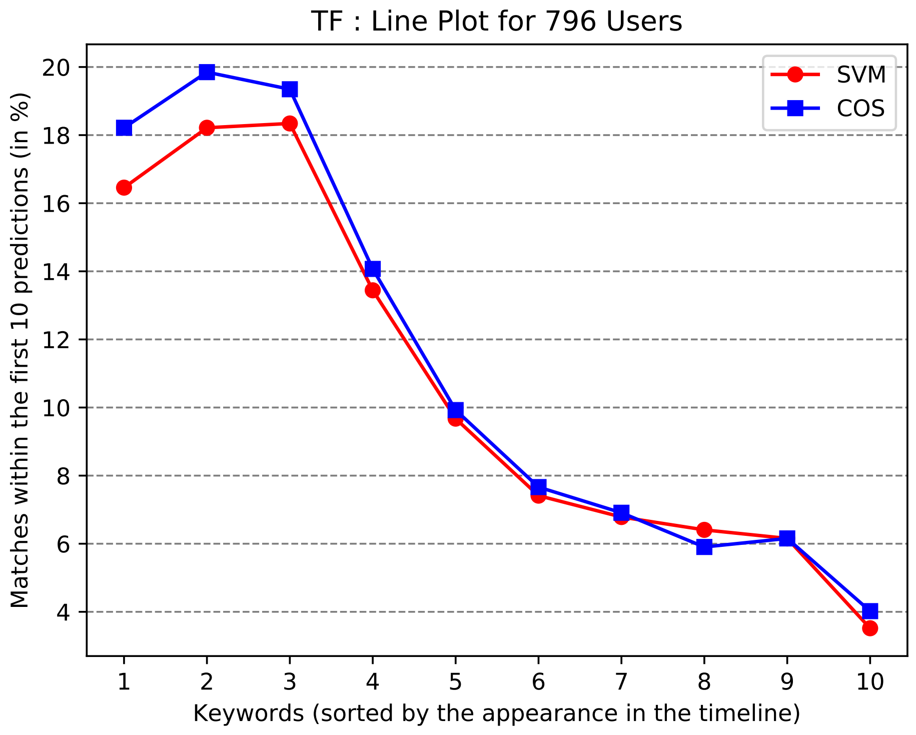


Figure 4.7.: This plot shows how the methods for learning the user profiles, COS and SVM performed relative to each other, while TF was used to generate the predictions. X-axis: Keyword to match (sorted by the appearance in the timeline) Y-axis: Amount of keywords, that were matched by BttFS within the first ten results (in %)

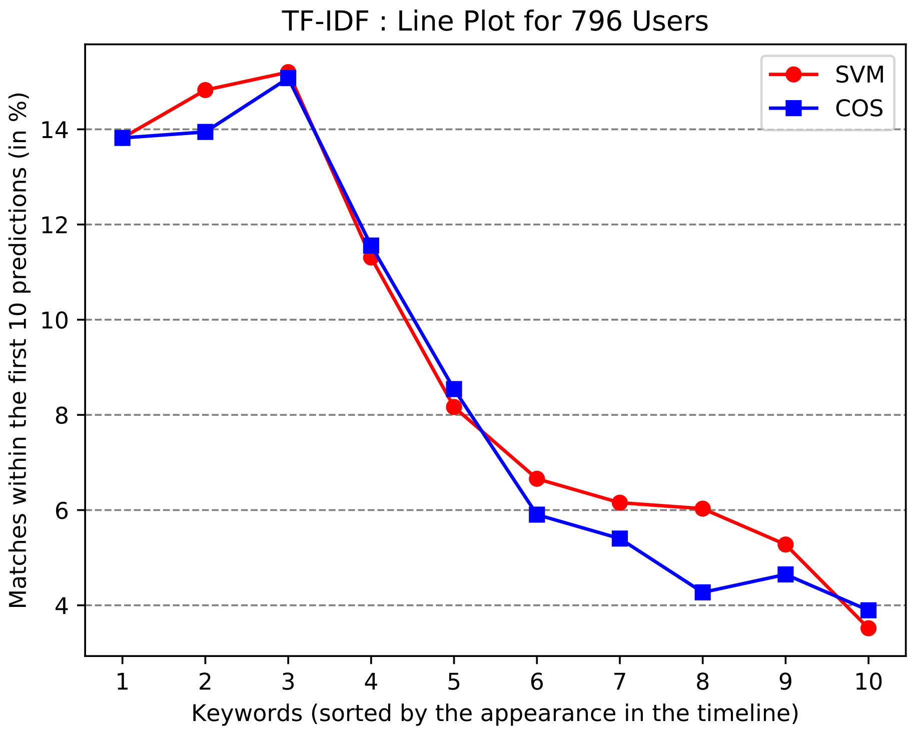


Figure 4.8.: This plot shows how the methods for learning the user profiles, COS and SVM performed relative to each other, while TF-IDF was used to generate the predictions. X-axis: Keyword to match (sorted by the appearance in the timeline) Y-axis: Amount of keywords, that were matched by BttFS within the first ten results (in %)

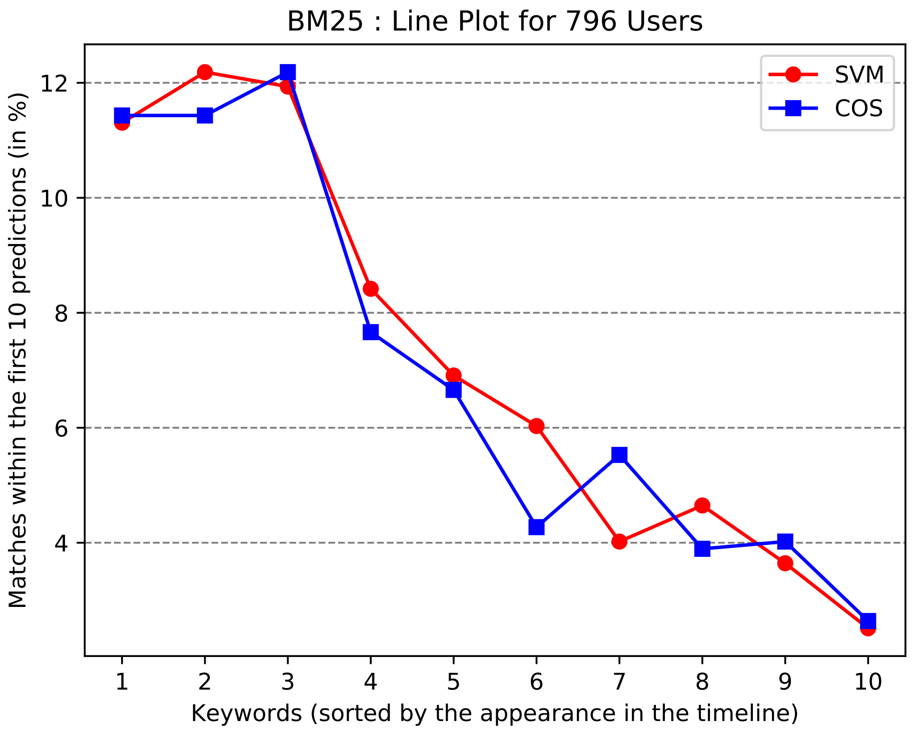


Figure 4.9.: This plot shows how the methods for learning the user profiles, COS and SVM performed relative to each other, while BM25 was used to generate the predictions. X-axis: Keyword to mach (sorted by the appearance in the timeline) Y-axis: Amount of keywords, that were matched by BttFS within the first ten results (in %)

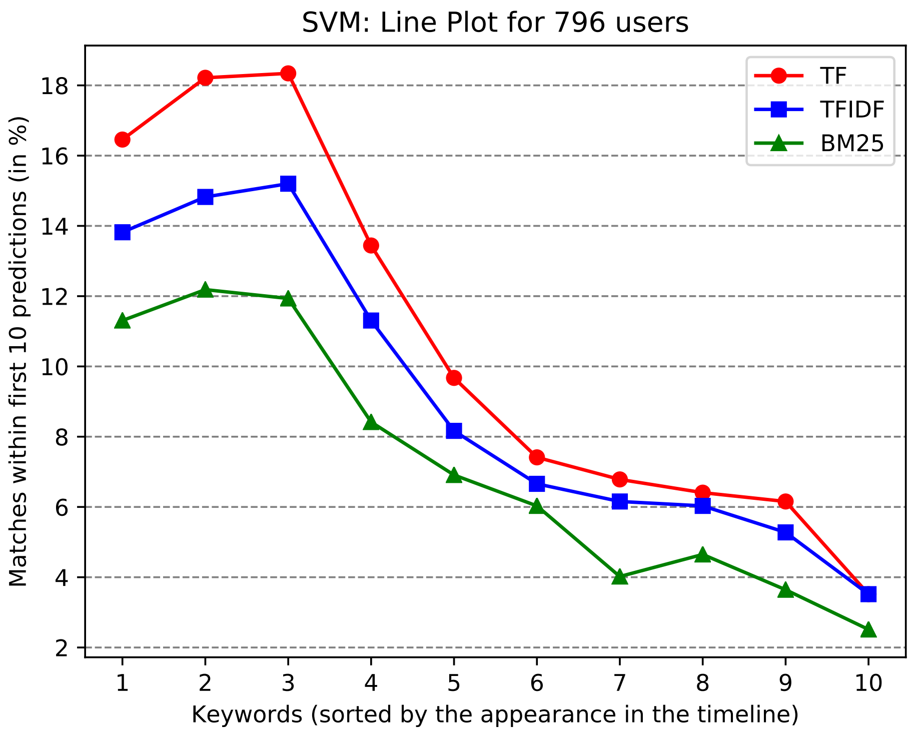


Figure 4.10.: This plot shows the results for the IR methods TF, TF-IDF, and BM25. The user profiles were learned with the SVM method. X-axis: Keyword to match (sorted by the appearance in the timeline) Y-axis: Amount of keywords, that was matched by BttFS within the first ten results (in %)

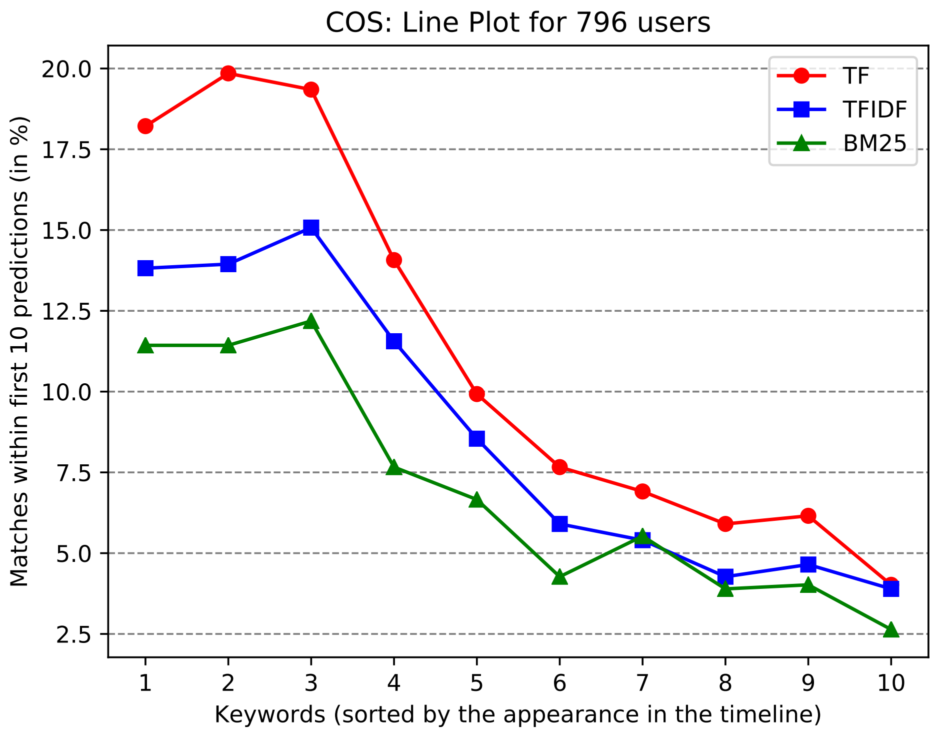


Figure 4.11.: This plot shows the results for the IR methods TF, TF-IDF, and BM25. The user profiles were learned with the COS method. X-axis: Keyword to match (sorted by the appearance in the timeline) Y-axis: Amount of keywords, that was matched by BttFS within the first ten results (in %)

4.4. BttFS Usability Test

To evaluate the usability of BTTFS, a friendly user test with four volunteers was carried out. The test was set up in a private environment where each participant had a time frame of 45 minutes to resolve all given tasks. The first task was to choose a topic of interest on which the participant‘s knowledge finding process would be based. In the second task, the participants had to create a user-profile in BTTFS. To create this profile it was mandatory to provide five keywords that explained the context of their information need on the previously chosen search topic. While the context-based search profile was active in BTTFS the participants browsed the web for 20 minutes to find information on their topic of interest.

In addition, every volunteer had to mark five web-pages in BTTFS as important, which provided useful information for the chosen topic. In the next step, the crawl function of BTTFS was used on the marked pages. The crawl function, on the one hand, provided information to the user regarding useful linked pages and on the other hand to extend BTTFS’s user profile. After all previous tasks were completed, the participants had to use the ”create search phrase” functionality of BTTFS multiple times. For every search phrase provided by BTTFS the participants had to evaluate the first three results shown by the search engine, in order to determine if the suggested search phrase yielded relevant or irrelevant information regarding the participant’s search context.

After the experiment was done, each volunteer had to answer four questions:

1. Did BTTFS's functionalities assist your search?
2. Would you use BTTFS on a regular basis?
3. Was BTTFS easy to use?
4. Were the search-queries provided by BTTFS helpful for your information-finding process?

4.4.1. BttFS Usability Test Results

The answers to the first question "Did BTTFS's functionalities assist your search?" given by the study participants ranged from "clearly yes" to "not really". It could be seen that more advanced users were able to acquire knowledge on their search context faster and used assisting features of BTTFS like marking they keywords on the given web-page more often than the study participants which had a non-technical background. However, the participants that do not search the web on a regular basis felt highly assisted by BTTFS and the provided keyword suggestions.

The second question "Would you use BTTFS on a regular basis?" caused a spread of opinions under the participants. On the one hand, the participants mainly agreed, that BTTFS provided helpful features for their knowledge generation process, on the other hand, most participants agreed that they would use BTTFS only for complex topics that would need a lot of research.

The question "Was BTTFS easy to use?" had a strong tendency to "yes". While the creation of the user profile and the context keywords felt not very intuitive for one person, all participants agreed, that the usage of BTTFS felt naturally while they were searching the web. The usage of functions like marking keywords, crawling important pages, viewing the statistics page of their search, as well as getting provided search phrases by the system seemed to fit seamlessly into the participant's search process.

The fourth questions "Were the search-queries provided by BTTFS helpful for your information-finding process?" again showed a spread of opinions. Two participants were "slightly" and "really pleased" with the provided search-queries of BTTFS, while the other two participants ranged from "neither/nor" helpful to "not really helpful". While some search contexts allowed BTTFS to provide good keyword suggestions, the search-context of one user seemed to push BTTFS into one direction, which ended in very similar keyword suggestions due to the narrow context.

From the participants' feedback, it can be distributed, that BTTFS can assist the information finding process. However, BTTFS could be more efficient in some cases. The creating of search phrases regarding the user's context could be adopted, so already found or viewed pages could be excluded from the search, in order to have no redundancy in the users' search process and the data that BTTFS provides. Another problem that could be identified regarding BTTFS's usability was, that more advanced profiles, in terms of big data and large stored text amounts, can lead to bad performance for generating the search query for the user.

5. Discussion

As the results of the first scenario showed (Section 4.2), BTTFS was able to predict the exact search queries with a word amount ranging from one to five with a best accuracy between 9.38% to 1.61%. The whole set of results for the first scenario can also be viewed in Table 5.1. While a 9.38% matching-rate for single word queries seems to be a decent value, search phrases with five words got only matched occasionally by BTTFS. One reason for this can be found in fact, that BTTFS was optimized for context-based search. Therefore, the learning of a user profile and the extraction of search queries tend to rank, or weight query terms heavier when associated with the user's context. The used dataset, however, did not assure context-based web-searches for the whole user-session, and therefore the created results tend to be worse than hoped for. Another reason, why BTTFS tends to underperform for search phrases containing more than three words, can be found in the small learning dataset that was used when learning the profiles for each user session. Due to the diverse form of the AOL-Dataset regarding the different length of the user sessions, only a bare minimum of data for learning a user profile was used by BTTFS. Otherwise too many user-sessions would have been sorted out by the test-system for not meeting the minimal set of requirements. Therefore, both algorithms for learning a user profile would potentially perform better in the long run if more training and test

Query - phrase	TF & COS	TF-IDF & COS	BM25 & COS	TF & SVM	TF-IDF & SVM	BM25 & SVM
length = 1	9.38	8.98	6.25	8.98	8.98	5.86
length = 2	5.91	5.51	3.35	5.71	5.51	3.94
length = 3	3.02	1.95	1.42	2.49	2.49	1.07
length = 4	1.65	0.47	0.24	1.18	0.47	0.24
length = 5	1.29	0	0	1.61	0.32	0

Table 5.1.: This table shows the exact results for all methods used for the 1st scenario. (All values stated in %)

data would be available like in a real-term scenario, however, to proof this point more data has to be gathered and analyzed in additional studies.

These considerations are also valid for the results that were shown in the second scenario that was discussed in Section 4.3. However, the combination of the COS learning function and the TF ranking function yielded promising results ranging between 18.22% - 19.85% matching rate on average, for the first three single word queries that appeared in advancing order on the timeline of the user actions. Regarding these results, BTTFS predicted the second future search word a user would type in 19.85% of all observations and the third future keyword a user would type in 19.35% of all cases.

The whole set of results for the second scenario can be viewed in Table 5.2. When examining the results, a trend can be seen that keywords that are closer to the present are predicted better by BTTFS than keywords that appear further in the future of a user session. However, the general performance of BTTFS for the second scenario showed promising results. With further optimization of the learning functions and additional weighting functions even better results could be yielded.

When considering the central question of this thesis:

"Is it possible to anticipate a user's future information need by exploiting the past browsing behavior regarding a defined context of information need?"

The results of both scenarios tend to enforce an affirmative answer to this raised question. BTTFS was able to predict the future user behavior correctly in some of the given cases and accelerated in performance when only single keywords or low-length search phrases had to be matched. Therefore it can be stated, that a future information-need of a particular user can in many cases be derived from the past browsing behavior when the context of the information need stays constant. The possibility to switch between different profiles (search contexts) seems to aid the user in order to get helpful search-term suggestions by BTTFS for the particular context. Therefore, it can be stated that BTTFS can aid the user in an information generation process.

Keyword	TF & COS	TF-IDF & COS	BM25 & COS	TF & SVM	TF-IDF & SVM	BM25 & SVM
#1	18.22	13.82	11.43	16.46	13.82	11.31
#2	19.85	13.94	11.43	18.22	14.82	12.19
#3	19.35	15.08	12.19	18.34	15.20	11.93
#4	14.07	11.56	7.66	13.44	11.31	8.42
#5	9.92	8.54	6.66	9.67	8.17	6.91
#6	7.66	5.90	4.27	7.41	6.66	6.03
#7	6.91	5.40	5.53	6.78	6.16	4.02
#8	5.90	4.27	3.89	6.41	6.03	4.65
#9	6.16	4.65	4.02	6.16	5.28	3.64
#10	4.02	3.89	2.64	3.52	3.52	2.51

Table 5.2.: This table shows the exact results for all methods used for the 2nd scenario. (All values stated in %)

6. Conclusion

This thesis was built around the context-based Web-Information-Agent BTTFS. Besides all the features BTTFS provides to aid the user's information finding and re-finding process, the primary focus of this thesis was centered on the automatic search phrase creation for the user's context, that BTTFS provides. In more detail, the different methods for learning a user profile as well as the different weighting schemes, which were used in BTTFS's automated search phrase creating-process, were evaluated by their overall performance and their performance relative to each other. This generated knowledge was used to answer the central question of this thesis:

"Is it possible to anticipate a user's future information need by exploiting the past browsing behavior regarding a defined context of information need?"

6.1. What has been done in this thesis?

In the first part of chapter 2, the weighting schemes, and the machine learning approaches, that are used by BttFS to create automated search queries for the user's context to assist the user's information finding process, were presented. The weighting functions TF, TF-IDF, and BM25 were discussed in more detail and commonly used standard formulas of these functions were presented. Furthermore, it was discussed how BTTFS learns and expands a user profile with a cosine-similarity method or a one-class Support Vector Machine when only positive relevance feedback was used to keep BTTFS's usability on a high level.

In the second part of chapter 2, three prior Web-Agents namely: WebACE, SurfAgent, and WebMate were described, to provide an overview of their architecture, functionalities, and the methods that were used for learning a user profile and aiding the user's information finding process. Furthermore, in the third part of chapter 2 selection of one-class classification methods like the k-centers method the k-nearest neighbor method, the one-class Gaussian model, and the k-means clustering.

In chapter 3, BTTFS's architecture, as well as its main functionalities were pointed out. BTTFS consists of two main parts. A WebExtension component that handles the user-interactions and acts as the interface to the Web, and a native python application component that holds the main logic for learning and expanding user profiles as well as the creation of search phrases for the user's context if demanded. However, besides the context-based search phrase creation, BTTFS also aids the user with multiple functionalities for finding and re-finding information. For example, BTTFS allows the user to mark the keywords, which define the context of a profile, on Web-pages for faster recognition. Furthermore, BTTFS allows the user to mark Web-pages as important, crawl sub-links of essential Web-pages, get suggestions for pages that could be interesting for the user's context. As well as to view a profile own statistics page that includes high-value information about the personal browsing behavior and a personalized browsing-history for the context of the active profile.

In chapter 4, the AOL-dataset was explained, which was used to test the precision of BTTFS's search phrase creation functionality in two different scenarios. Beforehand, the AOL-dataset was adjusted, to meet minimum requirements per user-session, for BTTFS to have enough data for each user to learn a user profile and to have enough data left to perform the prediction tests. Therefore, the AOL-dataset was reduced to 796 individual user-sessions, which were tested in an exact query-phrase matching scenario and a single query-word matching scenario. For the exact query-phrase scenario, it was tested how well BTTFS could predict exact query-phrases the users's entered into a search engine when only the prior browsing behavior of a user was learned. For the single query-word matching scenario, it was tested how well BTTFS could predict single keywords within the users query-phrases after the prior browsing behavior was learned. Furthermore, the chapter 4 also includes a friendly user test, which was carried out to test the usability of BTTFS. Therefore, four volunteers had to perform different tasks, to test and various aspects of BTTFS in real scenarios.

In chapter 5, the results for both of the mentioned scenarios were analyzed and interpreted. In general, it can be stated, that BTTFS generated the best results for both scenarios when COS was used for learning a user profile and TF was applied as the weighting function. The exact search-phrase matching scenario did yield decent results for phrases with a word count between one and three but underperformed for search-phrases with a word count of four and five. However, this could be explained by the fact, that the used AOL-dataset does not assure one single context per user-session, while BTTFS was optimized for the use within one particular context. The single query-word matching scenario, however, yielded promising results. With the combination of COS and TF, the first three keywords of the timeline of 796 tested users were correctly predicted between 18% to 20% on average.

Furthermore, the results showed, that the closer to the present a keyword of the timeline appeared the more likely the keyword was predicted correctly by BTTFS.

6.2. What can be concluded from this thesis?

When considering the performance regarding the 796 tested user-session of the AOL-dataset, the results showed that the SVM method for learning a user profile was slightly outperformed by the COS method. However, the difference between the two methods turned out to be insignificant in some cases. Moreover, the results showed that the TF weighting function seemed to yield the best results for both scenarios. Therefore, it can be stated that the COS method for learning a user profile, in combination with TF as the chosen weighting function produced the most promising results.

The results also showed that BTTFS's performs superiorly for predicting single keywords correctly, then for predicting whole search phrases. While the scenario where BTTFS was tested for matching single keywords in a user-session yielded better results as expected, BTTFS seemed to under-perform in the scenario, were whole search phrases should be matched.

When considering the central question of this thesis: *"Is it possible to anticipate a user's future information need by exploiting the past browsing behavior regarding a defined context of information need?"*

The results that were acquired in this thesis tend toward an affirmative answer to this question since BTTFS was able to predict the future user behavior correctly in some of the given cases and accelerated in performance when only single keywords or low length search phrases had to be matched. Therefore, when the context of the information need of a user stays constant, future information need can be derived from prior browsing behavior.

7. Future Work

The results gained in this thesis showed that BTTFS was able to predict future search phrases of many tested users-sessions to a certain degree. However, the used AOL-dataset did not assure a particular context per user-session which naturally could lead to slightly biased results, since BTTFS was developed with the primary goal to run with one specified context per created user-profile. Therefore, experiments with a dataset that only contains context-specific data per user could aid to determine BTTFS potential for matching future query phrases and single keywords of users, based on their prior browsing behavior, in an even more accurate way.

Furthermore, BTTFS showed the possibility of accelerating a user's information generation process, by assisting the user with automated search queries that can be stated as relevant for the user's context. Work could be invested, to implement even more advanced techniques for learning a user profile and weighting the keywords, to gain even more precision on the automatic generated search queries.

Another way to improve BTTFS in the future could be based on more customized methods for every single user. Due to the user-related relevance feedback and context defining keywords, which are needed by BTTFS, it is in no way assured that BTTFS can obtain excellent results for every specific user. If the relevance feedback of a user is broad and vague, the density and accuracy of the user's context could be weakened, which could lead to automated search queries, generated by BTTFS, that are not satisfying for a user. Therefore, an expansion of BTTFS could be planned for automatic context keyword creation to define a user's context automatically. Furthermore, an alternative could be found for replacing the need of BTTFS for relevance feedback of a user, with an own system component, to minimize relevance feedback that tends to very subjective for each user.

8. Acknowledgements

First and foremost, I am grateful to my supervisor Dipl-Ing. Dr.techn. Kern Roman, who gave me inspiration for this research project and provided me with constant support and advice.

Furthermore, I want to thank Corinna Bargehr, Moritz Lipp and Clemens Lauermann, who supported me emotionally and intellectually in the previous years and months.

Last but not least, I want to thank my mother, father, and brother for their support threw all these years, who always brought me back on track when I was losing my course.

Bibliography

Anuradha, J et al. (2015). "A brief introduction on Big Data 5Vs characteristics and Hadoop technology." In: *Procedia computer science* 48, pp. 319–324 (cit. on p. 2).

AOL (2017). *AOL data-set*. URL: http://jeffhuang.com/search_query_logs.html (cit. on pp. iii, 37, 39).

Bhatia, Sumit, Debapriyo Majumdar, and Prasenjit Mitra (2011). "Query Suggestions in the Absence of Query Logs." In: *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '11. Beijing, China: ACM, pp. 795–804. ISBN: 978-1-4503-0757-4. DOI: 10.1145/2009916.2010023. URL: http://doi.acm.org/10.1145/2009916.2010023 (cit. on p. 2).

Bishop, Christopher M (1995). *Neural networks for pattern recognition*. Oxford university press (cit. on p. 15).

Boley, Daniel (1998). "Principal direction divisive partitioning." In: *Data mining and knowledge discovery* 2.4, pp. 325–344 (cit. on p. 19).

Boley, Daniel et al. (1999). "Document Categorization and Query Generation on the World Wide Web Using WebACE." In: *Artificial Intelligence Review*, pp. 365–391.

Chang, Chih-Chung and Chih-Jen Lin (2011). "LIBSVM: a library for support vector machines." In: *ACM transactions on intelligent systems and technology (TIST)* 2.3, p. 27 (cit. on p. 12).

Charikar, Moses et al. (2004). "Incremental clustering and dynamic information retrieval." In: *SIAM Journal on Computing* 33.6, pp. 1417–1440 (cit. on p. 21).

Chen, Liren and Katia Sycara (1998). "WebMate: A Personal Agent for Browsing and Searching." In: *Proceedings of the Second International Conference on Autonomous Agents*. AGENTS '98. Minneapolis, Minnesota, USA: ACM, pp. 132–139. ISBN: 0-89791-983-1. DOI: 10.1145/280765.280789. URL: http://doi.acm.org/10.1145/280765.280789 (cit. on pp. 24, 25).

Dreiseitl, Stephan et al. (2010). "Outlier detection with one-class SVMs: an application to melanoma prognosis." In: *AMIA Annual Symposium Proceedings*. Vol. 2010. American Medical Informatics Association, p. 172 (cit. on p. 12).

Forbs (2018). *Web statistics*. URL: https://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#72a5ceaa17b1 (cit. on p. 2).

Gauch, Susan and Robert P Futrelle (1994). "Experiments in automatic word class and word sense identification for information retrieval." In: *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval*, pp. 425–434 (cit. on p. 25).

Google (2018). *Chrome Extensions*. URL: https://developer.chrome.com/extensions (cit. on p. 28).

Han, Eui-Hong, Daniel Boley, et al. (1998). "WebACE: A Web Agent for Document Categorization and Exploration." In: *Proceedings of the Second International Conference on Autonomous Agents*. AGENTS '98. Minneapolis, Minnesota, USA: ACM, pp. 408–415. ISBN: 0-89791-983-1. DOI: 10.1145/

280765.280872. URL: http://doi.acm.org/10.1145/280765.280872 (cit. on pp. 18, 19).

Han, Eui-Hong, George Karypis, et al. (1997). "Clustering based on association rule hypergraphs." In: *DMKD*, p. 0 (cit. on p. 18).

internetLiveStats (2018). *Amount of Searchqueries*. URL: http://www.internetlivestats.com (cit. on pp. 1, 4).

Kramer, Samuel Noah (1963). *The Sumerians: Their history, culture, and character*. University of Chicago Press (cit. on p. 7).

MDN (2017). *WebExtensions MDN*. URL: https://developer.mozilla.org/de/Add-ons/WebExtensions (cit. on p. 28).

MDN (2018). *WebExtensions APIs*. URL: https://developer.mozilla.org/de/Add-ons/WebExtensions (cit. on p. 28).

Miller, George A (1995). "WordNet: a lexical database for English." In: *Communications of the ACM* 38.11, pp. 39–41 (cit. on p. 35).

Muller, K-R et al. (2001). "An introduction to kernel-based learning algorithms." In: *IEEE transactions on neural networks* 12.2, pp. 181–201 (cit. on p. 12).

New, What's (2018). *JumpStation*. URL: https://web.archive.org/web/20010620073530/http://archive.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/old-whats-new/whats-new-1293.html (cit. on p. 17).

NLTK (2018). *NLTK*. URL: http://www.nltk.org/ (cit. on p. 35).

P. Deutsch, A.Emtage and A.Marine (2018). *Archie, FTP*. URL: http://archie.icm.edu.pl/archie-adv_eng.html (cit. on p. 17).

Robertson, Stephen (2004). "Understanding inverse document frequency: on theoretical arguments for IDF." In: *Journal of documentation* 60.5, pp. 503–520 (cit. on p. 9).

Rosenfeld, Roni (1994). "Adaptive statistical language modeling: A maximum entropy approach." In: (cit. on p. 25).

Salton, Gerard and Christopher Buckley (1988). “Term-weighting approaches in automatic text retrieval.” In: *Information processing & management* 24.5, pp. 513–523 (cit. on pp. 8, 9).

Salton, Gerard, Anita Wong, and Chung-Shu Yang (1975). “A vector space model for automatic indexing.” In: *Communications of the ACM* 18.11, pp. 613–620 (cit. on p. 10).

Sanderson, Mark (2010). “Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press 2008. ISBN-13 978-0-521-86571-5, xxi+ 482 pages.” In: *Natural Language Engineering* 16.1 (cit. on p. 9).

Schütze, Hinrich, Christopher D Manning, and Prabhakar Raghavan (2007). *An introduction to information retrieval*. Cambridge University Press, (cit. on p. 8).

SCIKit (2018). *SCIKit One Sided SVM*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html (cit. on p. 38).

SciKitLearn (2018). *one-classSVM*. URL: http://scikit-learn.org/stable/auto_examples/svm/plot_oneclass.html (cit. on p. 12).

Singhal, Amit et al. (2001). “Modern information retrieval: A brief overview.” In: *IEEE Data Eng. Bull.* 24.4, pp. 35–43 (cit. on pp. 7, 10).

Somlo, Gabriel L and Adele E Howe (2001). “Incremental clustering for profile maintenance in information gathering web agents.” In: *Proceedings of the fifth international conference on Autonomous agents*. ACM, pp. 262–269 (cit. on pp. 21, 22).

Somlo, Gabriel L. and Adele E. Howe (2003). “Using Web Helper Agent Profiles in Query Generation.” In: *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS ’03. Melbourne, Australia: ACM, pp. 812–818. ISBN: 1-58113-683-8. DOI:

10.1145/860575.860706. URL: http://doi.acm.org/10.1145/860575.860706 (cit. on p. 22).

Soup, Beautiful (2018). *Beautiful Soup Documentation*. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (cit. on p. 34).

Stradling, James (2018). *Unsupervised Machine Learning with One-class Support Vector Machines*. URL: https://thisdata.com/blog/unsupervised-machine-learning-with-one-class-support-vector-machines/ (cit. on pp. 12, 38).

Tarassenko, Lionel et al. (1995). "Novelty detection for the identification of masses in mammograms." In: (cit. on p. 15).

Tax, David Martinus Johannes (2001). "One-class classification." In: (cit. on pp. 13–16).

Venkataraman, Ganesh et al. (2016). "Instant Search: A Hands-on Tutorial." In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '16. Pisa, Italy: ACM, pp. 1211–1214. ISBN: 978-1-4503-4069-4. DOI: 10.1145/2911451.2914806. URL: http://doi.acm.org/10.1145/2911451.2914806 (cit. on p. 2).

WikiBM25 (2018). *Wikipeida-BM25*. URL: https://en.wikipedia.org/wiki/Okapi_BM25#cite_note-1 (cit. on p. 9).

WikiCoSDistance (2018). *Wikipeida-CoSDistance*. URL: https://en.wikipedia.org/wiki/Cosine_similarity (cit. on pp. 10, 11).

WordStream (2018). *History of Searchengines*. URL: https://www.wordstream.com/articles/internet-search-engines-history (cit. on p. 17).

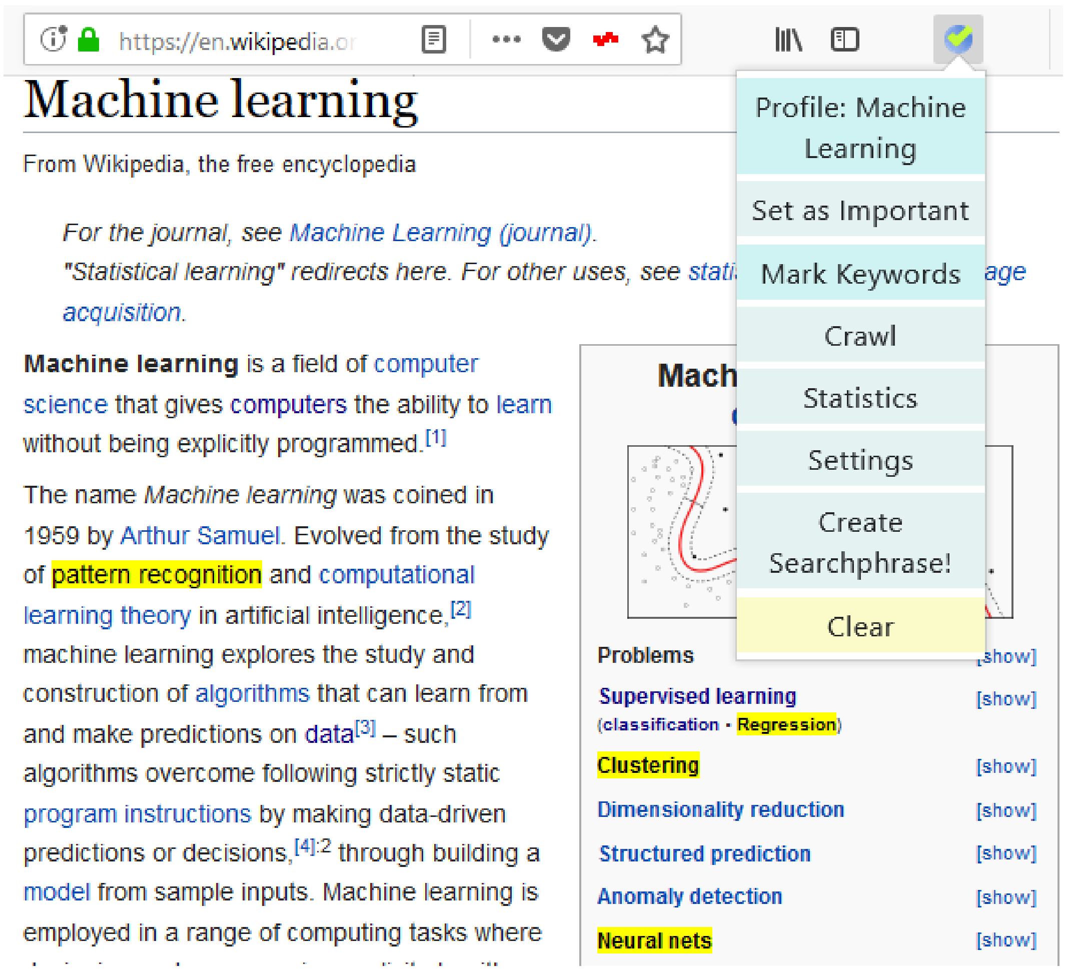
Yousef, Malik, Naim Najami, and Waleed Khalifav (2010). "A comparison study between one-class and two-class machine learning for MicroRNA target detection." In: *Journal of Biomedical Science and Engineering* 3.03, p. 247 (cit. on pp. 14, 15).

Ypma, Alexander and Robert PW Duin (1998). “Support objects for domain approximation.” In: *ICANN 98*. Springer, pp. 719–724 (cit. on p. 14).

Appendix A.

Further Resources

This chapter includes further resources that were not considered in this thesis or appeared in a strongly shortened fashion concerning their length, however, could be interesting for the reader of this thesis.



Machine learning

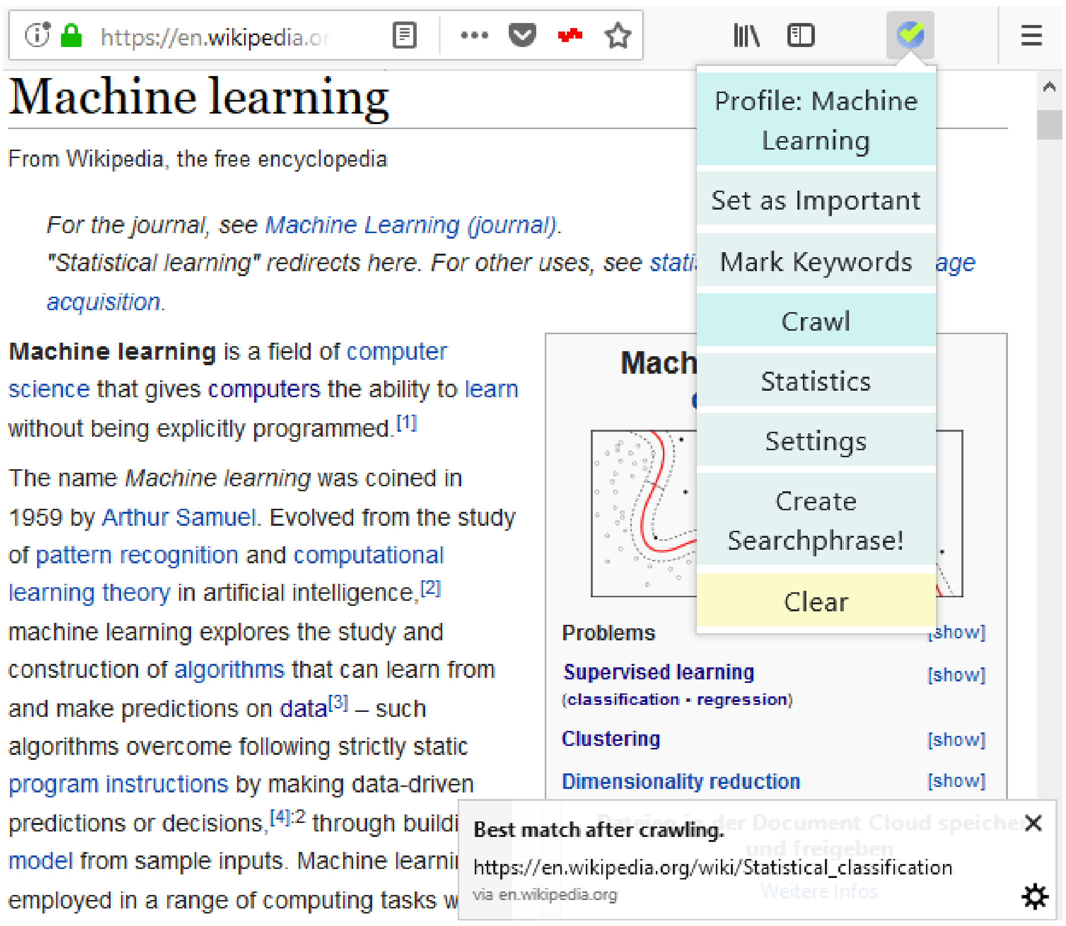
From Wikipedia, the free encyclopedia

For the journal, see Machine Learning (journal).
"Statistical learning" redirects here. For other uses, see stati… age acquisition.

Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.[1]

The name *Machine learning* was coined in 1959 by Arthur Samuel. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence,[2] machine learning explores the study and construction of algorithms that can learn from and make predictions on data[3] – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions,[4]:2 through building a model from sample inputs. Machine learning is employed in a range of computing tasks where

Figure A.1.: This Figure shows a screenshot - excpert of BttFS when the "Mark Keywords" option was used.



Machine learning

From Wikipedia, the free encyclopedia

For the journal, see Machine Learning (journal).
"Statistical learning" redirects here. For other uses, see stati age acquisition.

Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed.[1]

The name *Machine learning* was coined in 1959 by Arthur Samuel. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence,[2] machine learning explores the study and construction of algorithms that can learn from and make predictions on data[3] – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions,[4]:2 through buildi model from sample inputs. Machine learni employed in a range of computing tasks w

Best match after crawling.
https://en.wikipedia.org/wiki/Statistical_classification
via en.wikipedia.org

Figure A.2.: This Figure shows a screenshot - excpert of BttFS when the "Crawl" option was used. In the bottom right a notification of BttFS can be seen, where the best crawled web-page is shown regarding TF of the profile's keywords.

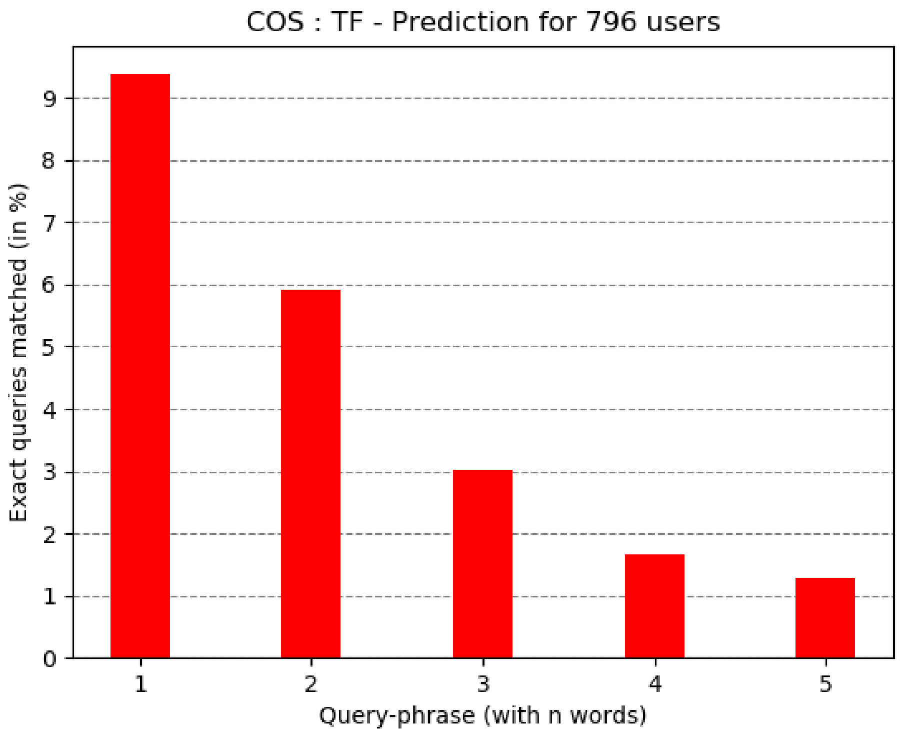


Figure A.3.: This plot shows how often an exact query-phrase was predicted correctly by BttFS. X-axis: Length in words of the exact query-phrases the users submitted to the search engine. Y-axis: Amount of exact query-phrase that was matched by BttFS (in %).

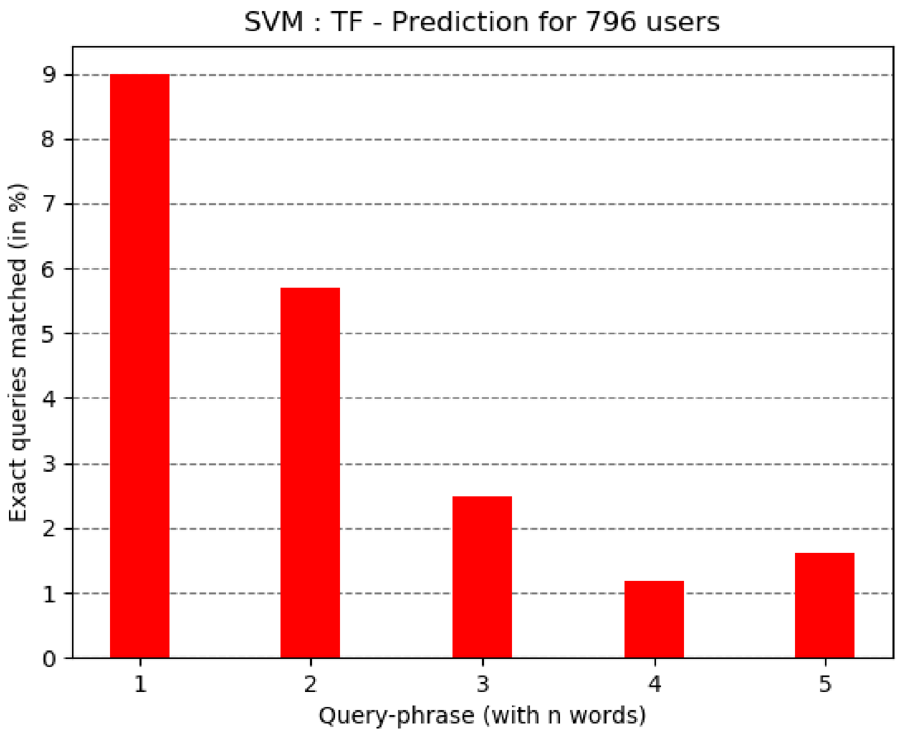


Figure A.4.: This plot shows how often an exact query-phrase was predicted correctly by BttFS. X-axis: Length in words of the exact query-phrases the users submitted to the search engine. Y-axis: Amount of exact query-phrase that was matched by BttFS (in %).

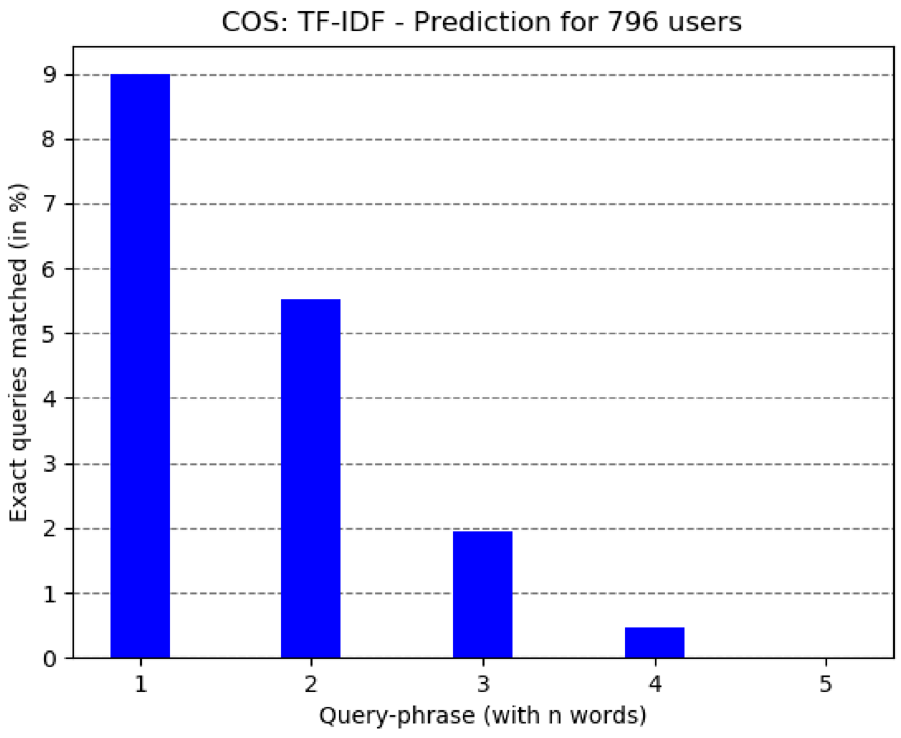


Figure A.5.: This plot shows how often an exact query-phrase was predicted correctly by BttFS. X-axis: Length in words of the exact query-phrases the users submitted to the search engine. Y-axis: Amount of exact query-phrase that was matched by BttFS (in %).

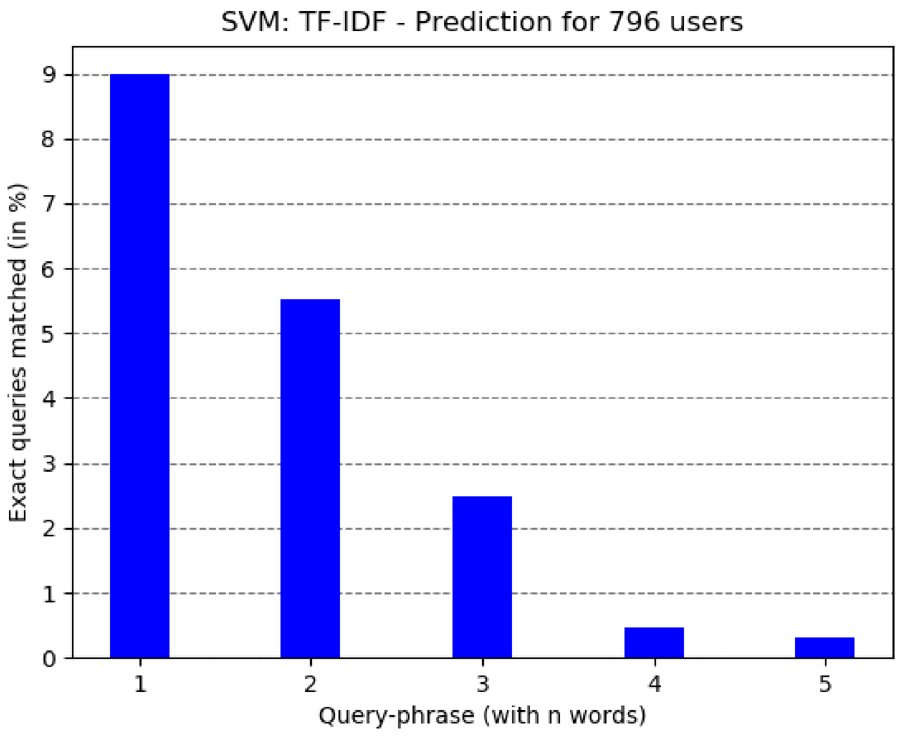


Figure A.6.: This plot shows how often an exact query-phrase was predicted correctly by BttFS. X-axis: Length in words of the exact query-phrases the users submitted to the search engine. Y-axis: Amount of exact query-phrase that was matched by BttFS (in %).

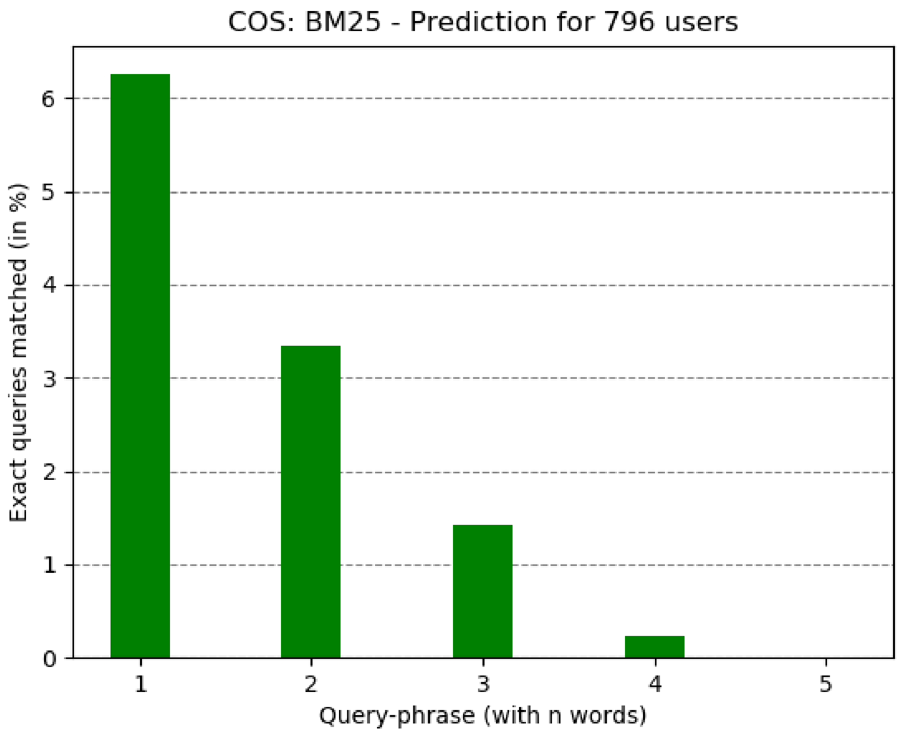


Figure A.7.: This plot shows how often an exact query-phrase was predicted correctly by BttFS. X-axis: Length in words of the exact query-phrases the users submitted to the search engine. Y-axis: Amount of exact query-phrase that was matched by BttFS (in %).

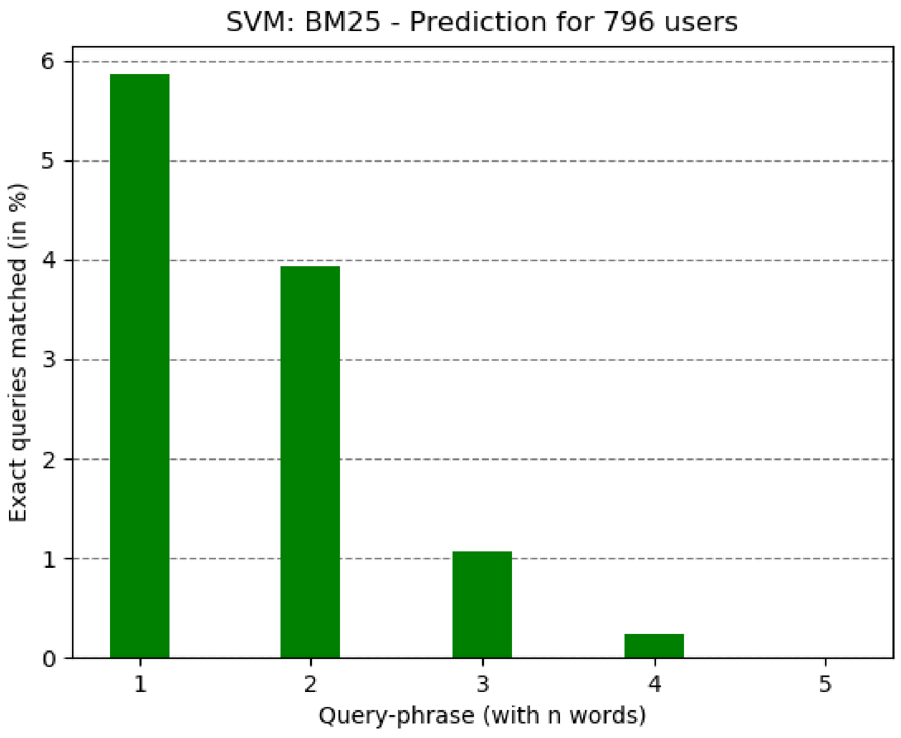


Figure A.8.: This plot shows how often an exact query-phrase was predicted correctly by BttFS. X-axis: Length in words of the exact query-phrases the users submitted to the search engine. Y-axis: Amount of exact query-phrase that was matched by BttFS (in %).

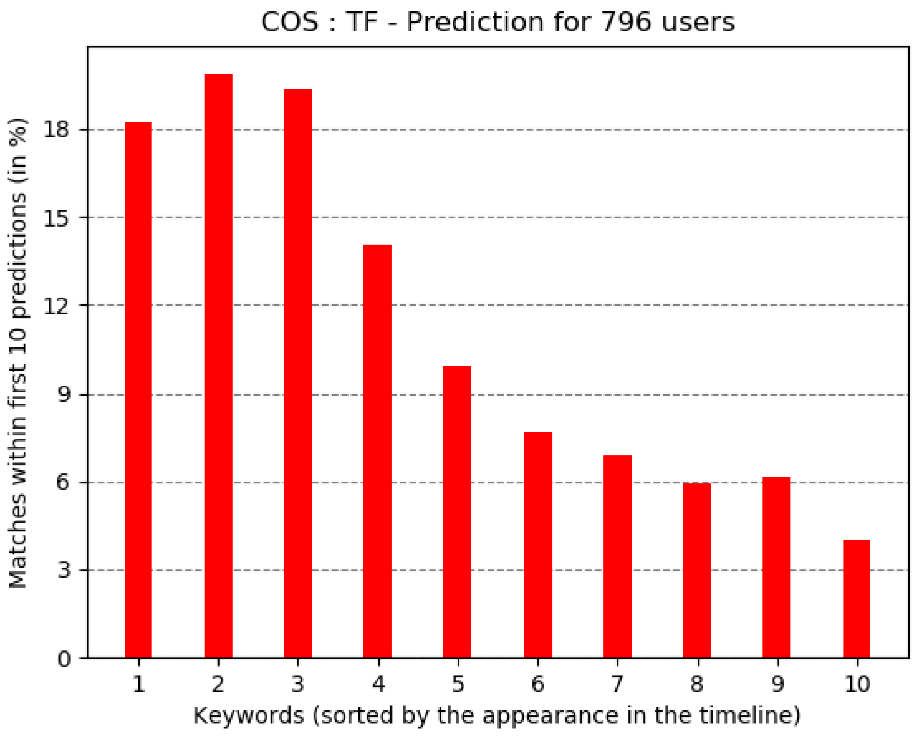


Figure A.9.: This plot shows how often an single query-keyword was predicted correctly by BttFS. X-axis: Keyword to match (sorted by the appearance in the timeline) Y-axis: Amount of keywords, that was matched by BttFS within the first ten results (in %).

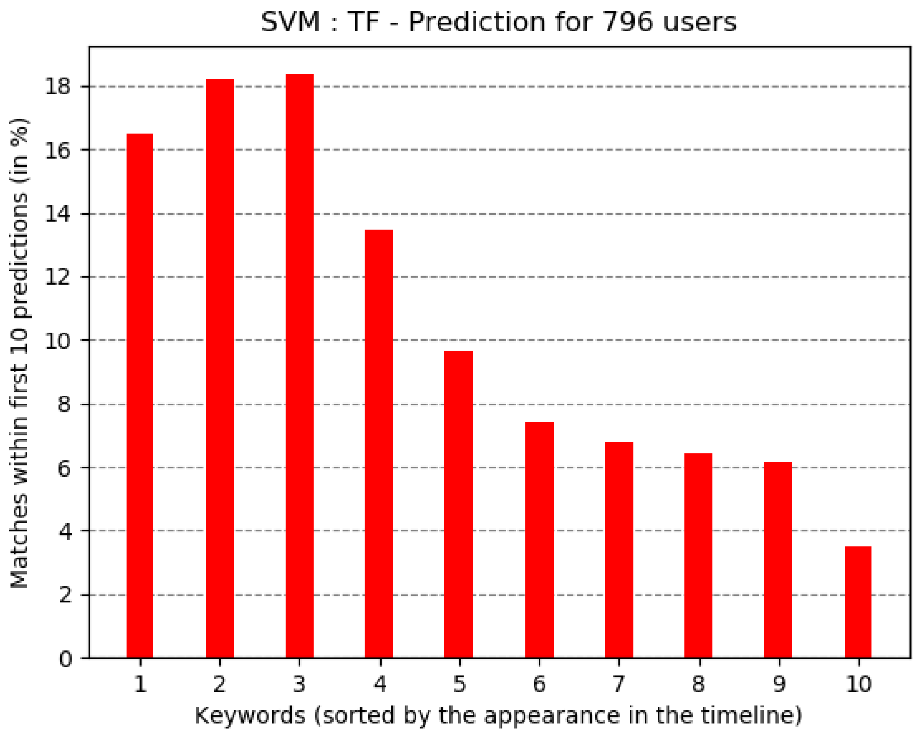


Figure A.10.: This plot shows how often an single query-keyword was predicted correctly by BttFS. X-axis: Keyword to match (sorted by the appearance in the timeline) Y-axis: Amount of keywords, that was matched by BttFS within the first ten results (in %).

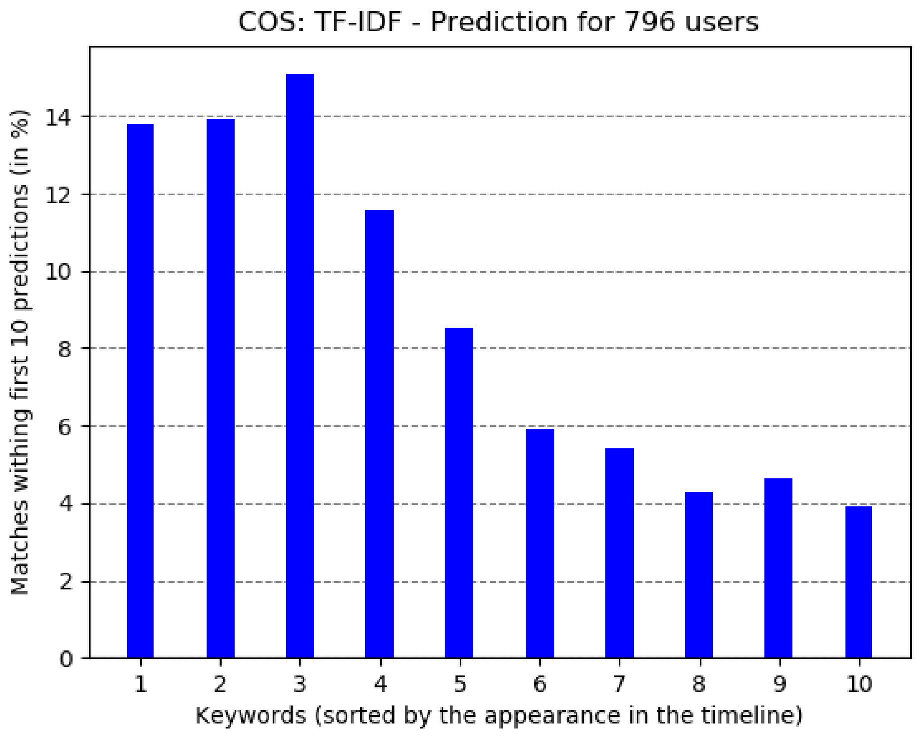


Figure A.11.: This plot shows how often an single query-keyword was predicted correctly by BttFS. X-axis: Keyword to match (sorted by the appearance in the timeline) Y-axis: Amount of keywords, that was matched by BttFS within the first ten results (in %).

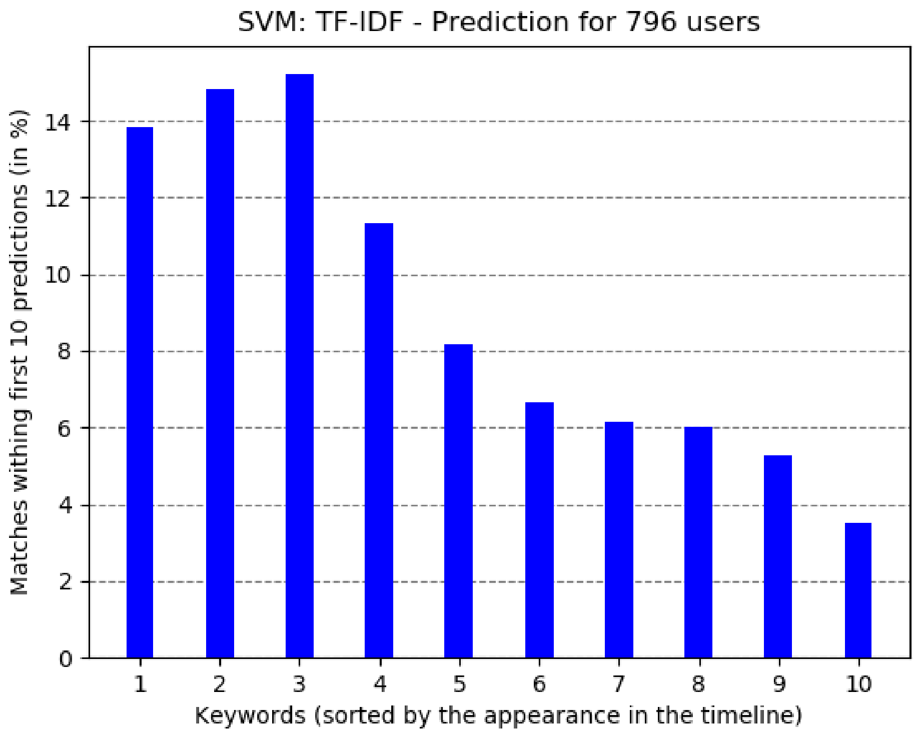


Figure A.12.: This plot shows how often an single query-keyword was predicted correctly by BttFS. X-axis: Keyword to mach (sorted by the appearance in the timeline) Y-axis: Amount of keywords, that were matched by BttFS within the first ten results (in %).

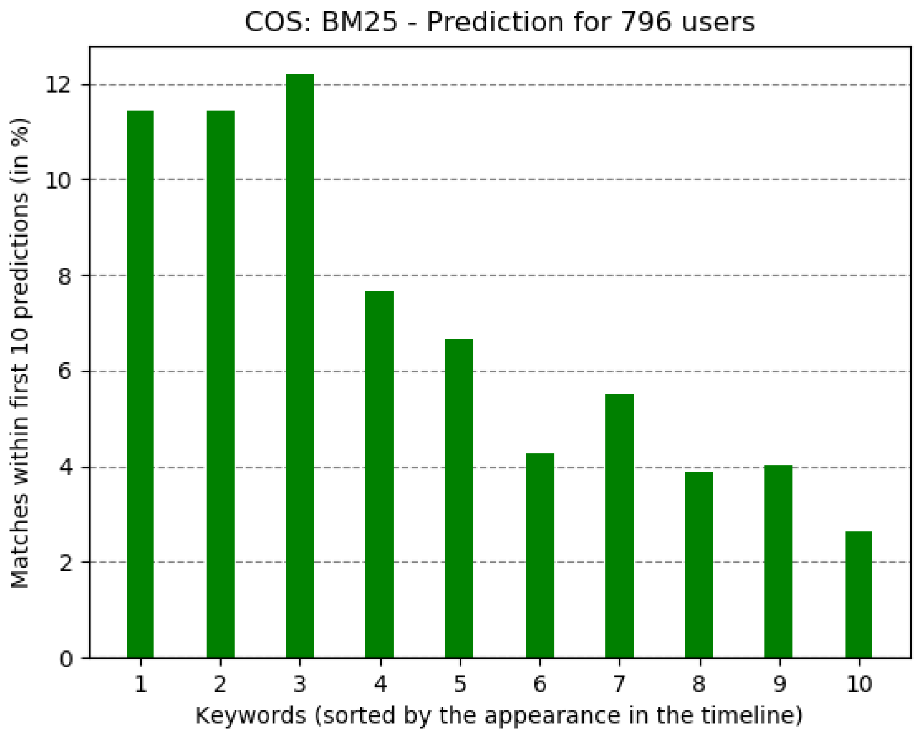


Figure A.13.: This plot shows how often an single query-keyword was predicted correctly by BttFS. X-axis: Keyword to match (sorted by the appearance in the timeline) Y-axis: Amount of keywords, that was matched by BttFS within the first ten results (in %).

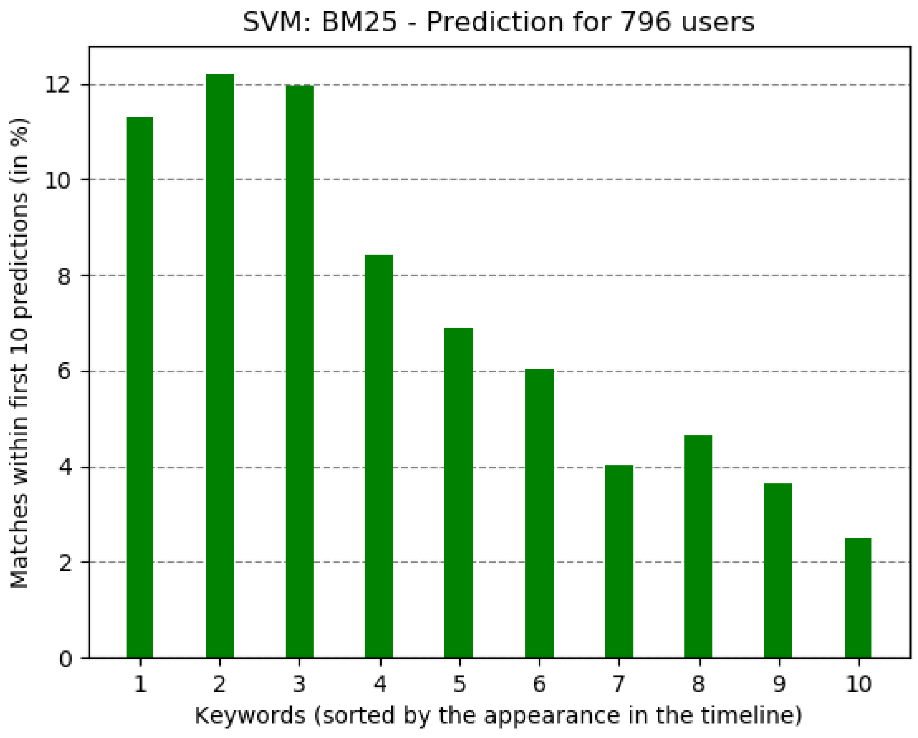


Figure A.14.: This plot shows how often an single query-keyword was predicted correctly by BttFS. X-axis: Keyword to match (sorted by the appearance in the timeline) Y-axis: Amount of keywords, that was matched by BttFS within the first ten results (in %).