Insert here your thesis' task.

Czech Technical University in Prague
Faculty of Information Technology
Department of Knowledge Engineering

Master's thesis

Text Signals Relevance Improvement for Full Text Search

Bc. Jan Hnízdil

Supervisor: Ing. Jan Šedivý, PhD.

5th May 2015

Acknowledgements

I am using this opportunity to express my gratitude to everyone who supported me through this thesis, especially to my supervisor Jan Šedivý for his patience, guidance and advice during the project work.

I would also like to thank my family for their supportive approach regarding my thesis.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 5th May 2015 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Hnízdil, Jan. *Text Signals Relevance Improvement for Full Text Search.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2015.

Abstrakt

Ačkoliv se vyhledávání informací na webu stalo standardem a často oblíbeným zdrojem pro hledání informací již před mnoha lety, úloha hledání relevance dokumentů k danému uživatelskému dotazu má stále mnoho slabých míst, které je zapotřebí zlepšit. Tato práce se snaží nalézt takové textové příznaky, které by zlepšily výsledky full-textového vyhledávání, a tím i spokojenost uživatelů, za využití datasetů od společnosti Seznam.cz. Za prvé jsou v rámci této diplomové práce analyzovány hlavní LTR algoritmy, evaluační míry a běžně používané textové signály známé z literatury. Za druhé byl navržen a naimplementován systém pro testování a evaluaci nově přidaných textových signálů a nakonec byly tyto nově přidané signály porovnány s anonymizovanými signály, které v současnosti používá Seznam.cz, prostřednictvím velké sady experimentů.

Klíčová slova Učení se řadit, AdaRank, Seznam.cz, Full-text, vyhledávání, příznaky

Abstract

Although web search has become a standard and often favored source of information finding many years ago, the task of searching relevance documents to given user query has still a lot of weak spaces need to be improved. This thesis is trying to find new text relevance signals to improve full-text search and user satisfaction via datasets provided by Seznam.cz. First of all, there is analyzed and evaluated major LTR algorithms, evaluation metrics and commonly used text signals known from literature. Second, system for testing and evaluation of new signals was designed and implemented and finally bunch of experiments over the new text signals were conducted and results were compared with anonymized baseline signals provided by Seznam.cz.

Keywords Learning to Rank, AdaRank, Seznam.cz, Full-text, search, text signals

Contents

List of Figures

List of Tables

Introduction

As recently as the 1990s, studies showed that most people on the Internet preferred getting information from other people rather than from information retrieval systems. [1] Nevertheless at these times peoples also preferred to use human travel agents to book their travel or to use paper train schedule, so this fact was not surprising at all. However, information retrieval has undergone many changes and optimizations so now people are satisfied with results of web search most of the time. Web search has become a standard and often favored source of information finding. Actually nowadays search is still one of the most important applications for Internet users at all.

Nevertheless the task of searching relevance documents to given user query has still a lot of weak spaces need to be improved. To successfully solve the problem, machine learning models have to be applied and good signals for relevancy ranking algorithms have to be found.

This whole area is called information retrieval (IR) that in general is the activity of obtaining information resources relevant to an information need from a huge collection of information resources. Moreover, to fully cover user needs is required to sort the relevant document by its relevancy. We can assume that if the most relevant document will be at the top of the search, user will be the most satisfied as well.

Searches can be based on meta-data or on full-text [1] (or other content-based) indexing. An information retrieval process begins when a user enters a query into the system. Queries are formal statements of information needs, for example search strings in web search engines.

Schema of the search engines for Information retrieval

The task of the search starts when a user identifies his need for information, then it involves searching and locating of an information resource and it ends when the information is retrieved and delivered to the user in a demanded form and the information need of the user is eventually satisfied. As shown

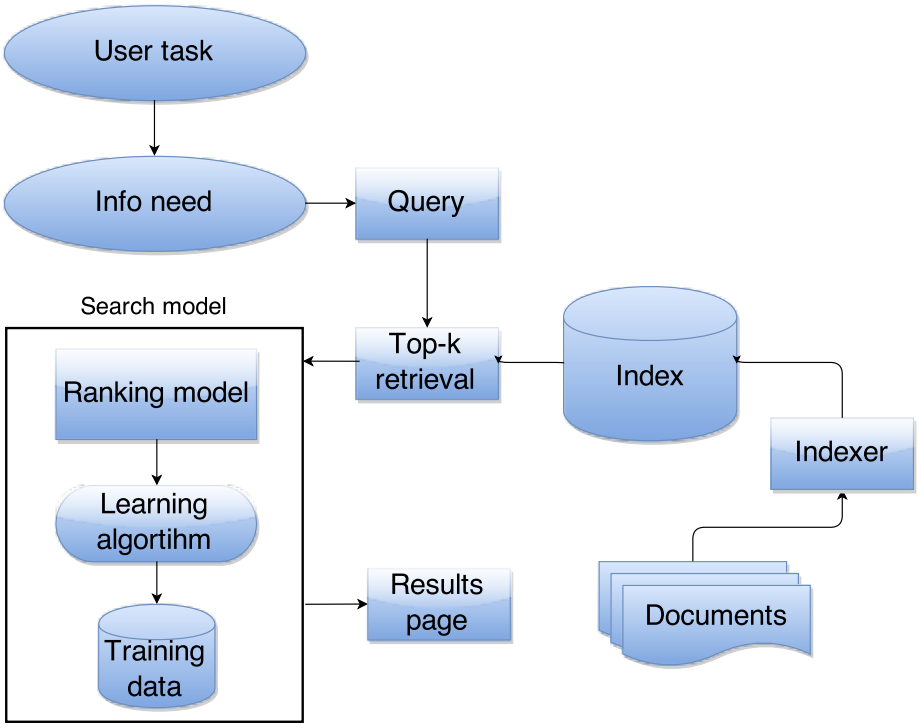


Figure 0.1: Traditional Information Retrieval model

on the Figure 0.1 - nowadays, the modern IR deals with data storage, analysis and retrieval of documents, algorithms etc. and it is a very sophisticated process. In this work, we focus only on the part of search model, that means to ranking model, learning algorithms and especially training model which covers different types of signals used to train the models for more relevant result pages.

Goals

This work follows several milestones. Complete assignment is attached to the beginning of this work so we are going to mention just the key goals here.

First and the most importantly we are trying to find new text relevance signals (features) improvement for full-text search, that means we want to find such signals, which would be able to help with better learning to sort the search models in terms of user satisfaction. Text signals are mentioned across all this work; from research and analysis in Chapter 1 to experiments in Chapter 4. Second we want to create system for testing and evaluation of new signals so there will be possible to test arbitrary new signals and compare it with

the other ones. These part of work is described mainly in Chapter 2 and 3. To create such a system includes a selection of Learning to Rank algorithms, evaluation measure and different types of preprocessing phases. Therefore another no less important goal is to make research about LTR algorithms, evaluation metrics, etc. which is included primarily in Chapter 1.

Chapter 1

State-of-the-art

Although this work tries to find the best text signals, it is necessary to use proper algorithms and evaluation metrics as well.

First this chapter is going to review and analyze major Learning to Rank algorithms, which are the keys to sort set of web documents by a document relevance, second, analyze and describe evaluation metrics for relevancy ranking, that make it possible to evaluate and compare relevance sorting results and finally review the commonly used text signals known from literature.

1.1 Learning to Rank algorithms

Learning to Rank (LTR) [2] is a relatively new field in which machine learning can be effectively applied to solve the task of creating a ranking model in Information Retrieval (IR). It helps solving IR problems such as document retrieval, collaborative filtering, sentiment analysis, computational advertising etc. LTR method aims at learning a model that given a query and a set of candidate documents finds the appropriate ranking of documents according to their relevancy.

LTR is a supervised machine learning method. A typical setting in learning to rank is that feature vectors describing a query-document pair are constructed and relevance judgments of the documents to the query are available. Given a training dataset of queries, documents and evaluations of how relevant the documents are, a LTR algorithm constructs a ranking model. The ranking model is then usually used to assign ranking scores to a new set of documents with unknown relevance. The ranking scores are finally used to order the given documents. The evaluation of the model's performance can be accomplished by a chosen performance measure.

There are plenty of algorithms available to solve Learning to Rank problems. The differences among them are not often significant because some of them differ only in loss functions (e.g. while one algorithm uses logistic loss function, the other algorithm uses hinge loss function).

1.1.1 General framework for Learning to Rank

The process of learning to rank can be described as follows. A training set is typically based on query-document pair. The necessity for using the query-document pairs as the training samples comes from the fact, that many features are based on the relation between query and document.

Training samples with relevance labels (with respect to a given query), a particular evaluation measure and eventually a validation dataset come as an input to the LTR algorithm. The algorithm uses the training dataset to construct a model which is then used to sort a set of testing samples and the ranking performance of the model is then evaluated by given performance measure. The aim of the learning is generally minimization of a loss function, or eventually maximization of a training performance measure.

More theoretically, let $Q = \{q_1, q_2, \ldots, q_n\}$ is set of queries, where n denotes the number of queries and there is a set of documents:

$$d_i = \{d_1^i, d_2^i, \ldots, d_{m(q_i)}^i\}$$

associated with each of the queries q_i. Then there is also a list of labels $y_i = \{y_1^i, y_2^i, \ldots, y_{m(q_i)}^i\}$ connected with particular document-query pairs, where $m(q_i)$ denotes the number of documents given for the query q_i, y_j^i represents the label of the j^{th} document d_j^i of the i^{th} query q_i. A feature vector $\vec{x}_j^i \in X$ is specified for each query-document pair $(q_i, d_j^i), i = 1, 2, \ldots, n; j = 1, 2, \ldots, m(q_i)$. Finally, we can define training dataset as a set:

$$S_{train} = \{(q_i, d_i, y_i)\}_{i=1}^n$$

The objective of learning is to create a ranking function $f : X \mapsto \Re$, such that for each query the elements in its corresponding document list can be assigned relevance scores using the function and then be ranked according to the scores. Specifically, we create a permutation of integers $\pi(q_i, d_i, f)$ for query q_i, the corresponding list of documents d_i, and the ranking function f.

The learning process turns out to be that of minimizing the loss function which represents the disagreement between the permutation $\pi(q_i, d_i, f)$ and the list of ranks y_i, for all of the queries.

1.1.2 Categorization of Learning to rank algorithms

There are several approaches to learn the Learning to rank algorithms. According to the [2] and talks at many leading conferences the LR algorithms can be categorized into three groups by their input representation and loss function.

This section will slightly describe all the three category and will show main representatives of these categories. The biggest attention will be focused on list-wise algorithm AdaRank (see 1.1.2.6) since it will be chosen for implementation and further extension and experimentation.

Point-wise approach

Point-wise approach handles the problem by transforming ranking into regression or classification of single objects. The model then takes only one sample at a time and either it predicts its relevance score or it classifies the sample into one of the relevancy classes (e.g. a class of slightly relevant documents).

In this case it is assumed that each query-document pair in the training data has a numerical or ordinal score. Then LTR problem can be approximated by a regression problem given a single query-document pair, predict its score. A number of existing supervised machine learning algorithms can be readily used for this purpose. Ordinal regression and classification algorithms can also be used in point-wise approach when they are used to predict score of a single query-document pair, and it takes a small, finite number of values.

1.1.2.1 Random forest

Random forests are an ensemble learning method for classification, regression and other tasks, developed by Breiman and Cutler [3], that operate by constructing a multitude of decision or regression trees at training time and outputting the class that is the mode of the classes (this is for classification) or mean prediction (this is for regression) of the individual decision or regression trees so each of the trees in the ensemble votes for the output value. The final output is then determined by all the trees in the ensemble.

Bagging is used to reduce the correlation between each pair of random trees in the ensemble. Unlike single decision or regression trees which are likely to suffer from high Variance or high Bias (depending on how they are tuned) Random Forests use averaging to find a natural balance between the two extremes.

1.1.2.2 Rc-Rank

RC-Rank is an algorithm provided by Seznam.cz. Since not all the details are publicly available, we can provide only a brief basic description of RC-Rank. RC-Rank belongs to the category of methods applying point-wise approach. To the best of our knowledge and according to [4] , the algorithm works on the similar basis as *MART* algorithm, i.e. it is a boosting algorithm that is using the idea of multiple additive trees. However, the major difference is in the type of decision trees that are used by the algorithm. While *MART* uses *regression trees*, RC-Rank makes use of *oblivious decision trees*.

1.1.2.3 MART

MART (Multiple Additive Regression Trees, a.k.a. Gradient boosted regression tree) [5] is an approach utilizing a boosted tree model in which the output of the model is a linear combination of the outputs of a set of regression trees.

MART can be trained to minimize any general cost (classification, regression, ranking), however, underlying model upon which MART is build is the least squares regression tree.

Since MART belongs to the family of boosting algorithms, it runs a several rounds of boosting and in each step there is a regression tree added and its weight is determined. The final scoring (ranking) function is defined as follows in (1.1), where N denotes number of trees, α_i denotes weight of the i-th tree and $f_i(\vec{x_j})$ represents output from the tree to given input.

$$F_N(\vec{x_j}) = \sum_{i=1}^{N} \alpha_i f_i(\vec{x_j}), \tag{1.1}$$

Pair-wise approach

In this case Learning to rank problem is approximated by a classification problem learning a binary classifier that can tell which document is better in a given pair of documents. The goal is to minimize average number of inversions in ranking.

For each pair of documents, it returns a label determining relative relevance of the pair, whether the first document should be ranked above the second one or vice versa.

1.1.2.4 RankNet

RankNet, proposed by [6], employs a simple probabilistic cost function (relative entropy), as a loss function and gradient descent as an algorithm to train a neural network model. They use the idea of [7] (RankSVM) to train the model on pairs. However, the trained ranking function maps to reals, since it would be computationally slow to rank items on the pair basis. It means, that document pairs are used as learning instances but then only single documents are evaluated during the ranking process. The approach can be used with many underlying algorithms. [8] used neural networks because of its flexibility ([9] claims that two layer neural network can approximate any bounded continuous function) and efficiency in a test phase (compared to kernel methods).

Let (A, B) be a pair of samples, $\bar{P}_{AB}$ a target probability of sample A being ranked higher than sample B, $o_i = f(x_i)$ and $o_{ij} = f(x_i) - f(x_j)$, the cross entropy cost (loss) function is then defined in (1.2).

$$C_{ij} = C(o_{ij}) = -\bar{P}_{ij} - (1 - \bar{P}_{ij}) \log(1 - P_{ij}) \tag{1.2}$$

Mapping from the outputs to probabilities is provided by a logistic function (1.3).

$$P_{ij} = \frac{e^{o_{ij}}}{1 + e^{o_{ij}}} \tag{1.3}$$

And thus resulting function C_{ij} becomes

$$C_{ij} = -\bar{P}_{ij} o_{ij} + \log(1 + e^{o_{ij}}) \tag{1.4}$$

This cost function (slightly modified for the neural net purposes) is then optimized by the means of neural networks, i.e. back-propagation and forward-propagation. The ranking model is then represented by a vector of weights (parameters of a neural net) which have been learned.

Herbrich et al. [10] approach the problem as ordinal regression, i.e. learning the mapping of an input vector to a member of an ordered set of numerical ranks (intervals on real numbers). The loss function used in their method depends on pairs of examples and their target ranks. It is complicated to find the interval thresholds, though.

1.1.2.5 RankSVM

RankSVM (also called RankingSVM) is an application of Support vector machine proposed by [10], which is used to solve certain ranking problems.

It is another algorithm applying pair-wise method, classifying pairs of documents and determining their relative relevance. RankSVM approaches the ranking as ordinal regression and therefore the thresholds of the classes have to be trained as well. RankSVM employs minimization of hinge loss function. It also allows direct use of kernels for non-linearity. RankSVM was one of the first algorithms with pair-wise approach to the problem.

List-wise approach

List-wise algorithms are similar to pairwise, however they consider a list-wise structure of the ranking, trying to minimize some loss function which looks at the ordering of all query-document pairs and not just pairs.

List-wise handles the problem directly, by considering a whole document list as a learning sample. For example, by using all the relations among all the documents belonging to one particular query and not only by comparing pairs or single samples. The advantage is that the approach is natural and straightforward and it employs all information about the documents including their position in a particular list. The disadvantage is that it is challenging and complicated to optimize a function defined on the whole list.

1.1.2.6 AdaRank

AdaRank is a boosting algorithm based on AdaBoost method introduced by Jun Xu and Hang Li [11] in 2007. The algorithm repeatedly constructs so called 'weak rankers', trained on the re-weighted training data and finally linearly combines to make ranking predictions.

Boosting

Boosting (see figure 1.1) is a general technique for improving the accuracies of machine learning algorithms. The basic idea of boosting is to repeatedly construct 'weak learners' h_t at each round t by re-weighting over the training queries and form an ensemble of weak learners such that the total performance of the ensemble is 'boosted'. Freund and Schapire [12] have proposed the first well-known boosting algorithm called AdaBoost (Adaptive Boosting).

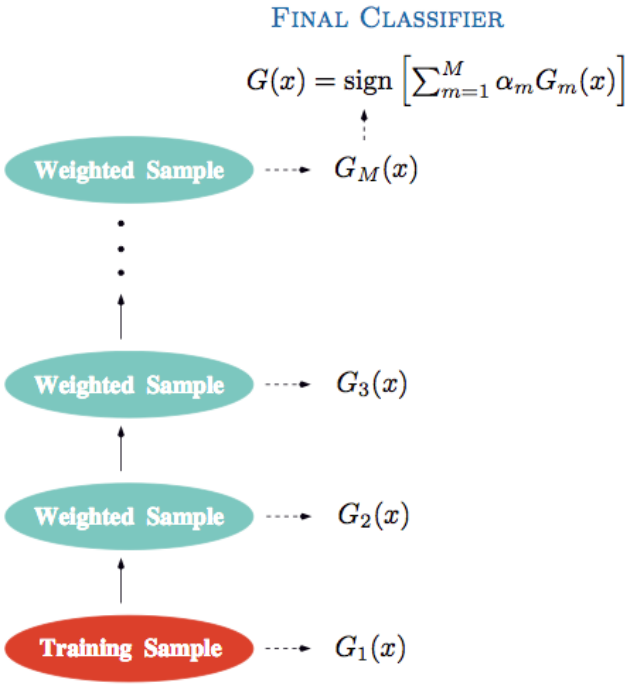


Figure 1.1: Boosting model

AdaRank is also one that tries to directly optimize multivariate performance measures, but is based on a different approach. AdaRank is unique in that it utilizes an exponential loss function based on IR performance measures and a boosting technique.

Algorithm

In this section, the algorithm AdaBoost, which can optimize a loss function based on the Information Retrieval performance measures, will be introduced.

The objective of the algorithm is to maximize the ranking performance measure on the training data (see Algorithm 1), where E represents any general performance measure (MAP, NDCG, WTA).

input : $S = \{(q_i, d_i, y_i)\}_{i=1}^{n}$, and parameters E and T;
output: Output ranking model: $f(\vec{x}) = f_T(\vec{x})$;
Initialize $P_1(i) = \frac{1}{m}$;
for $t = 1, \ldots, T$ **do**

- Create weak ranker h_t with weighted distribution P_t on training data S;
- Choose α_t

$$\alpha_t = \frac{1}{2} \ln \frac{\sum_{i=1}^{m} P_t(i)\{1 + E(\pi(q_i, d_i, h_t), y_i)\}}{\sum_{i=1}^{m} P_t(i)\{1 - E(\pi(q_i, d_i, h_t), y_i)\}}$$

- Create f_t

$$f_t(\vec{x}) = \sum_{i=1}^{m} \alpha_k h_k(\vec{x})$$

- Update P_{t+1}

$$P_{t+1}(i) = \frac{exp\{-E(\pi(q_i, d_i, f_t), y_i)\}}{\sum_{j=1}^{m} exp\{-E(\pi(q_j, d_j, f_t), y_j)\}}$$

end

Algorithm 1: AdaRank

AdaRank takes a training set $S = \{(q_i, d_i, y_i)\}_{i=1}^{m}$ as input and takes the performance measure E (in this case NDCG) and the number of iterations T as parameters. The algorithm runs T rounds and at each round it creates a weak ranker $h_t(t = 1, \ldots, T)$. Finally, it returns a ranking model f which is a linear combination of the weak rankers.

At each round, AdaRank maintains a distribution of weights over the queries in the training dataset which is denoted at round t as P_t and weight on the i^{th} training query q_i at round t as $P_t(i)$. In the very beginning, AdaRank sets equal weights to the queries. At each round, it increases the weights of those queries that are not ranked well by the model created so far, denoted as f_t. As a result, the learning at the next round will be focused on the creation of a weak ranker that can work better on the ranking of those 'hard' queries.

1.1.2.7 LambdaMART

LambdaMART [6] is a method combining two different approaches. Firstly MART (section 1.1.2.3) that is based on boosted regression trees, and LambdaRank that uses Neural nets, gradient descent method and the idea of λ's.

Lambdas (λ's) can be understood as rules defining how to change the ranks of documents in a ranked list in order to optimize the performance. This is

different from other approaches that just define how to change the ranks of documents based on the performance measure (which can be a problem using certain measures, e.g. WTA).

According to [4] the LambdaMART is considered as state-of-the-art LTR algorithm.

1.2 Evaluation metrics

In order to measure the quality of a search engine, some evaluation metrics are needed. Similarly to other machine learning problems, it is necessary to decide how the performance of the final model will be evaluated. In many machine learning methods, the objective function which is being optimized during a learning phase is the same as the final measure evaluating the resulting model. For example, Mean squared error (MSE) can be used in both cases, as an objective function during the training of a regression model and when the performance of the resulting model is being evaluated.

Unfortunately, Learning to Rank is not the same case. Since a LTR performance measure involves sorting and it is non-smooth, it cannot be differentiated and thus it is very challenging to optimize the measure directly. Only a very few algorithms actually optimize the performance measure directly. Therefore, it is important to distinguish between an objective function and a performance measure in LTR.

Furthermore, ranking web documents is very subjective task so it is almost impossible to correctly determine which document permutation is better than another. Let us take a look on the short example. Consider we have two different rankings of four documents $d_1, \ldots, d_4$. First ranking (document permutation) is defined as follows:

$$\pi_1 = ((d_1, 0); (d_3, 1); (d_4, 1); (d_2, 0))$$

and second one:

$$\pi_2 = ((d_1, 1); (d_3, 0); (d_4, 0); (d_2, 1))$$

Second position in the tuple denotes document relevance (0 means irrelevant, 1 means relevant). For the first sight we can consider the task as a very difficult even for human because we do not know if it is more important to have relevant document at the first position in the ranking list, however to have relevant document at the last position as well (π_1 case) or it is better to have relevant documents at second and third position (π_2 case).

Part of evaluation models can handle with multi-graded relevance, part of them can handle just with a binary relevancy. Nevertheless there are two main category of the evaluation (performance) models with different approaches.

1.2.1 Position model

Position-based model assumes that user interacts (clicks) with the document (URL) in the list under two conditions: first, it is *relevant* and second, it is examined, where the examination probability is dependent only on the position on the document in the ranked list (it is not influenced by any other document in the list). It means that it is more likely that the first document in the list will be clicked than the eleventh document because the probability of examination is much lower at 11^{th} position. The position model is implemented, for example, by NDCG or MAP measures.

1.2.1.1 Winner Takes All (WTA)

Very simple and clear evaluation measure is WTA measure which is defined as follows:

$$WTA(f; D, Y) = \begin{cases} 1 & : \text{the top document of the list is relevant} \\ 0 & : \text{the top document of the list is irrelevant} \end{cases}$$

where f is the ranking function, D is a set of documents and Y is a set of relevance labels corresponding to the documents in the set D. There are only two possible outcomes of WTA. Either it is 1 or 0. The value depends only on the document that is ranked as the very first document in the ranked list. If the first document is relevant, the value of WTA is 1. It is 0, otherwise.

1.2.1.2 NDCG

Normalized Discounted Cumulative Gain (NDCG) measures the performance of a recommendation system based on the multi-graded relevance of the recommended entities. It varies from 0.0 to 1.0, with 1.0 representing the ideal ranking of the document–query pair. NDCG has got following rules:

$$DCG(f, D, Y) = \sum_{i=1}^{m} G(y(d_{\pi_f(i)}))disc(i) \tag{1.5}$$

where G is a increasing function called the `gain function`, `disc` is a decreasing function called the `position discount function`, and π_f is the result of ranking list given ranking function f.

$$IDCG(f, D, Y) = \max_{\pi} \sum_{i=1}^{m} G(y(d_{\pi_f(i)}))disc(i) \tag{1.6}$$

$$NDCG(f, D, Y) = \frac{DCG(f, D, Y)}{IDCG(f, D, Y)} \tag{1.7}$$

The most common implementation of the `gain function` G is set to $G(z) = 2^z - 1$ and `discount function (disc)` is set to

$$disc(z) = \frac{1}{log_2(1+z)}$$

if $z \leq C$, and $disc(z) = 0$ if $z > C$ (C is a fixed integer).

1.2.2 Cascade model

The cascade model is an extension of the *position* model. The cascade model assumes a user scans through ranked search results in order, and for each document, evaluates whether the document satisfies the query, and if it does, stops the search.

Apart from *position* model, the probability of interaction with the document d_i also depends on the documents that have been ranked above d_i and the relevance grades of those documents. It means that if there is a perfect match on the first position of a retrieved list of documents, it's not fully important how relevant the document on the further positions are, because the needs of the user would be satisfied by the top ranked document and the probability of further examination is rapidly decreasing. On the other side, if there are not really relevant results at the top of the list then the importance of the ranking on further positions is increasing.

1.2.2.1 Expected Reciprocal Rank (ERR)

ERR is a state-of-the-art performance measure developed by [13]. First, we need to model how likely it is that a given document will satisfy a given user query. Let we have 6 different grades from 0 to 5, with 0 meaning irrelevant and 5 meaning highly relevant. These are translated into probabilities of the document satisfying the search by mapping a grade k to:

$$G(k) = \frac{2^k - 1}{2^{k_{max}}} \tag{1.8}$$

Expected reciprocal rank is just the expectation of the reciprocal of the position of a result at which a user stops. Suppose for a query q, a system returns a ranked list of K documents $d_1, \ldots, d_K$, where the probability that document k satisfies the user query is given by the transform of the editorial grade $G(k)$ (1.8) assigned to the query-document pair. If we let s be a random variable denoting the rank at which we stop, the metric is the expectation of $\frac{1}{s}$,

$$ERR := \sum_{k=1}^{K} \frac{1}{k} G(k) \prod_{i=1}^{k-1} (1 - G(i)). \tag{1.9}$$

Chapelle showed in [13] that ERR can be easily adjusted to be computed in $O(n)$, even thought a naive way how to compute ERR has complexicity of $O(n^2)$.

1.2.2.2 Seznam Rank (SR)

SR is a performance measure used by Seznam.cz company. The measure takes into account only the top 20 documents for each query, i.e. it is $SR@20$ by default (see 1.2.2.3). The performance score for each query is given by following equation:

$$SR = \min\left(1, \sum_{k=1}^{20} w^{pos}(k) \cdot w^{rel}(y\left(d_{\pi_f(k)}\right))\right), \tag{1.10}$$

where $w^{pos}(k)$ is the weight given by the position k (specifying that top document are more important than bottom documents), $y\left(d_{\pi_f(k)}\right)$ is a relevance grade of the document ranked at the k^{th} position and $w^{rel}(y)$ is the weight according to the relevance grade y. The values given by (1.10) are summed over the top 20 documents and the lower value is saved - either 1 or the result of the summation. The weights w^{pos} and w^{rel} are secret constants provided by Seznam.cz in range $< 0.0; 1.0 >$.

1.2.2.3 Top K documents (@k)

There are measures (e.g. ERR, NDCG) that can be computed based only on top k elements of the ranked list. This type of setting can be marked by *@k* at the name of the measure, specyfing that measure will be computed based just on the first k elements.

That makes sense for the particular tasks. For example, if we know the exact number of displayed ranked documents in advance so it is pointless to evaluate such documents are not visible at the result page.

1.3 Commonly used (text) signals

In this section we will discuss the state-of-the-art signals for full-text relevancy search. First of all we will define and describe main category of the full-text relevancy signals, and then we will focus just on the text signals for the rest of this work.

Unsurprisingly the signals (features) are very important part of the task of learning LTR algorithms with meaningful evaluation metric value (i.e. NDCG, ERR, etc.). The signals are kind of secret spices for a lot of research teams around the world, such as Google, Microsoft, Facebook or Seznam (in the Czech republic). Therefore it is very difficult to collect at least part of them.

The best opportunity how it is possible to get some intuition into the signals are public competitions provided by big companies.

For example Yahoo! Learning to Rank Challenge [14] gave to the competitors just an overview of the features released in these datasets. They could not gave specifics of how these features were computed (because of above-mentioned reasons), nevertheless they provided a high-level description instead, organized by feature type.

Within this research we have divided the features, according to [14], into the main following categories with very brief introduction, although we will analyze and experiment only with the text signals afterwards.

Web graph These features tries to determine the quality or the popularity of a document based on its connectivity in the web graph. A famous example is PageRank [15] introduced by Co-Founder of Google, which is basically based on the web graph propagation with the number of inlinks and outlinks. Other example could be features that include distance or propagation of a score from known good or bad documents.

Document statistics This type of features is dependent only on the documents itself. It includes some basic statistics of the document such as the number of words in various fields. This category also includes characteristics of the url, title or headers.

Document classifier Several classifiers which are applied to the document, such as language, main topic, quality, type of page (e.g. navigational). Main goal is to classify document in these fields and then the results compare with query result.

Query Features that help in characterizing the query type: number of terms, frequency of the query and its terms. These features are dependent only on the queries.

Text match The most important and the biggest type of features, which is also used in this work. Text match uses for one thing content from the document and for another thing content from the query and try to find a matches between them. The basic features are computed from different section of the document (title, body, headers or url).

The match score can be either just a counts (e.g. number of occurrences in the query, the number of missing query terms or the number of extra terms) or can be more complex such as BM25.

Finally, there are also included proximity features try to quantify how far in the document are the query terms, or how far is the whole document and query (e.g. in the cosine similarity).

Topical matching These features find similarity between query and document list at the topic level, instead of the aforementioned word level. This can be done for instance by classifying both the query and the document in a large topical taxonomy.

Click Click features are kind of different approach try to employ the user feedback, most importantly the clicked results. For a given query and document, there are computed different click probabilities (i.e. probability of click, first click, last click, long dwell time click, etc.) If the given query is rare, these clicks features can be computed using similar, but more frequent queries.

1.3.1 Text signals

Since the work deals just with text signals, we will describe just the text signals in the following section. That means only the signals that were computed either from document texts or query texts, or, especially from both at once.

1.3.1.1 Google text signals

Google is considered as a state-of-the-art among web search engines so there is an assumption that its signals for full-text relevance are state-of-the-art as well. Unfortunately there is not complete, or even confirmed list of features used by Google so a lot of researchers tried to predict at least some of them. According to [16], where it is published list of 200 potential relevance signals, we will describe the most promising and such that satisfy above-mentioned definition of text signals:

- *Query Token in Title Tag*
- *Query Token Appears in H Tag*
- *Query Token is Most Frequently Used Phrase in Document*
- *Content Length*
- *Query Token Density*

  Denotes how many words from document matched with query tokens divided document content length.

- *Latent Semantic Indexing Query Token in Content (LSI)*

  LSI keywords (see 1.3.1.4) help search engines extract appropriate meaning even from words with more than one meaning.

- *LSI Query Tokens in Title and Description Tags*
- *URL Length*

- *Query Token Appears in URL*

Notwithstanding the above list there is no doubt that Google also uses the following advanced techniques as a text signals.

1.3.1.2 Vector space model

Vector space model (VSM) is very often used for document and query representation in information retrieval and other tasks of natural language processing thanks to its simplicity. In this algebraic model there are documents or queries represent as points in n-dimensional space, where every dimension means one word from dictionary. That means we have K n-dimensional vectors $d_i = (w_1, w_2, \ldots, w_n)$ for K documents in corpus.

Values in the n-dimensional vector can be set to just binary values in order to presence word in document, or it can represents more complex information (e.g. TF (term frequency) - in this case we can call this model bag-of-words, TF-IDF (see 1.3.1.3)), etc.)

This document representation has obviously many drawbacks, first it does not take into account orders of document in this model , second there is a lot of zeros in the vector, because average document contains reasonably smaller number of words than size of dictionary, and finally it is very difficult to compare query and document in such a high dimensional space (that is phenomenon called curse of dimenzionality [1]).

Signals from VSM In order to create signal from the query and document representation in vector space model is needed arbitrary distance function to compute query - document similarity. Value of the signal then represent distance from the query and document. There is an assumption that closer vectors in VSM are more similar than farther ones.

Suitable distance function for this task is the cosine similarity since we want to measure angle between two vectors instead of Euclidean distance which would not work for different sized document and query. Cosine similarity $S(d, q)$ between document d and query q is defined as follows:

$$S(d, q) = \frac{d * q}{\|d\|\|q\|} \tag{1.11}$$

There is a lot of models based on VSM. In the same manner can be created signals from more sophisticated VSM model such a LSI or LDA. The only one condition on the same dimensionality of document and query is required to be done.

[1] Curse of dimenzionality refers to fact that with growing dimension grows exponentially length of the space in which the document are and therefore all documents are separated by the same distance.

1.3.1.3 TF-IDF

TF-IDF [1], short for term frequency-inverse document frequency is a weighting scheme assigns to term t a weight in document d given by:

$$tf\text{-}idf_{t,d} = tf_{t,d} * idf_t \tag{1.12}$$

where tf denotes term frequency and idf denotes inverse document frequency. The weight has following properties:

- highest value when t occurs many times within a small number of documents (thus lending high discriminating power to those documents);
- lower value when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal);
- lowest value when the term occurs in virtually all documents.

1.3.1.4 Latent semantic indexing

Latent Semantic Indexing [17] is a technique that projects queries and documents into a space with latent semantic dimensions. In the latent semantic space, a query and a document can have high cosine similarity even if they do not share any terms, that means LSI overcomes two of the most problematic constraints of classical bag-of-words VSM: (I) synonymy [2] and (II) polysemy [3]. Synonymy is often the cause of mismatches in the vocabulary used by the authors of documents and the users of information retrieval systems.

We can look at LSI as a similarity metric that is an alternative to word overlap measures like TF-IDF. The latent semantic space that we project into has fewer dimension than the original space. LSI is thus method for dimensionality reduction.

Latent semantic indexing is the application of a particular mathematical technique, called Singular Value Decomposition or SVD, to a word-by-document matrix. SVD (and hence LSI) is a least-squares method. The projection into the latent semantic space is chosen such that the representations in the original space are changed as little as possible when measured by the sum of the squares of the differences.

We can represent each dimension at the new vector space as a linear combination of the word from dictionary. For example dimension d_i could be represented as $(2.4 * \text{"}car\text{"} + 1.8 * \text{"}bus\text{"} + 1.3 * \text{"}motorbike\text{"} + \ldots)$ which would stand for a vehicle topic.

LSI assumes that there is some underlying or latent structure in word usage that is partially obscured by variability in word choice. A truncated

[2] syntactically different words that have similar meanings

[3] syntactically the same words that have more than one meaning

singular value decomposition (SVD) is used to estimate the structure in word usage across documents. Retrieval is then performed using the database of singular values and vectors obtained from the truncated SVD. Performance data shows that these statistically derived vectors are more robust indicators of meaning than individual terms.

1.3.1.5 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA), presented by David Blei et al. [18], is a generative model trying to automatically discover topics that documents contain. In more detail, LDA represents documents as mixtures of topics that produce words with certain probabilities. It assumes that documents are produced in the following manner that we decide on the number of words N according to Poisson distribution and choose topic mixture for the document according to Dirichlet distribution [19] (let us consider there are two different topics and we pick vehicle topic with $\frac{1}{4}$ probability and animal one with $\frac{3}{4}$ probability).

Assuming this generative model for a collection of documents, LDA then tries to backtrack from the documents to find a set of topics that are likely to have generated the collection. [20]

1.3.1.6 Semantic Hashing

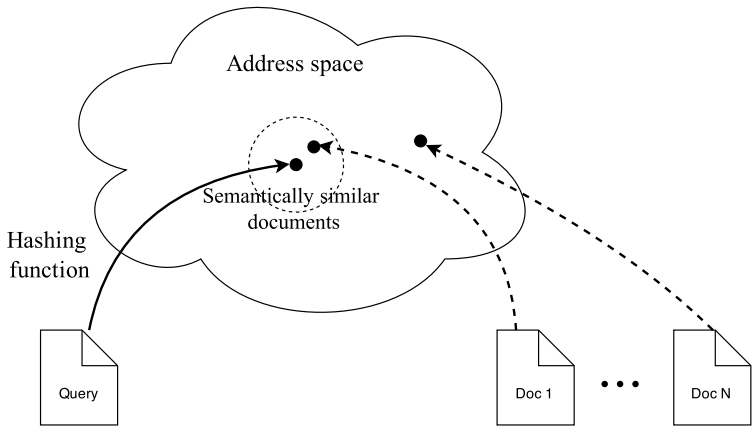


Figure 1.2: Semantic hashing

Semantic hashing, developed by G. Hinton researcher team [21], is another approach to create signals with latent (semantic) representations. Basic intuition about semantic hashing describes following Figure 1.2. Documents

are mapped into memory addresses in such a way that semantically similar documents are located at nearby addresses. Documents similar to a query document can then be found by simply accessing all the addresses that just slightly differ with define similarity function.

Semantic hashing is based on Deep learning algorithms, especially Stacked Denoising Autoencoders (SDA) (see 1.3.1.6), transform high dimensional word-count vector (for instance TF-IDF vector bag-of-words) to latent dimensional vector with lower number of dimension.

The main advantage of this approach is used nonlinear transformation, so there is not such restriction as in case LSI, which is linear method so it can only capture pairwise correlations between words.

Stacked Denoising Autoencoders (SDA) First for definition of SDA we have to describe principle of Autoencoders. Autoencoder [22] is a three-layer neuron network 1.3 trying to encode and then subsequently decode input through hidden (middle) layer to the same output as input. There is a one condition that input layer and output layer has to have got the same size.

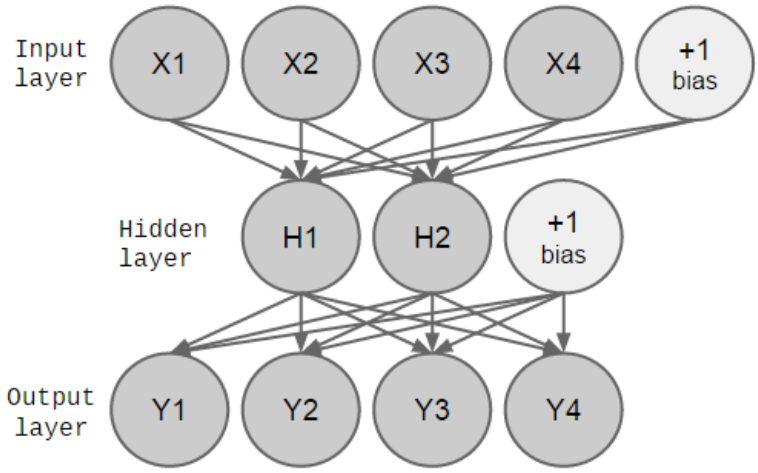


Figure 1.3: Model of Autoencoder

The key point of this approach is that the middle layer has less number of neurons than I/O layers so the neural nets has to find some hidden representation that does not loss any information during the encoding / decoding stage.

Second the idea behind "Denoising" Autoencoders (DA) is simple. In order to force the hidden layer to discover more robust features and prevent it from simply learning the identity, we train the Autoencoder to reconstruct the input from a corrupted version of it. That means the DA is a stochastic

version of the Autoencoder. Intuitively, a denoising Autoencoder does one extra thing in addition to classic Autoencoder: try to undo the effect of a corruption process stochastically applied to the input of the auto-encoder.

Finally Stacked Denoising Autoencoders is a neural network consisting of multiple layers of sparse Autoencoders in which the outputs of each layer is wired to the inputs of the successive layer. Following this manner we can create very deep neural network which is able to encrypt arbitrary input vector to meaningfully smaller dimension size.

1.3.1.7 Simhash

Simhash [23] is completely different approach from the Semantic Hashing (similar name is just a coincidence), which is currently commonly used to detect near-duplicate web documents during web crawling. Although the algorithm itself is based on hashing functions that are generally built so that identical files or blobs of data share the same hash and arbitrarily different blobs create entirely different signature, simhash tries to make different approach and it assumes that similar documents will have also similar hashes (that's why it is called simhash).

Principle of Simhash is essentially very simple. Given list of words from document $D = \{w_1, w_2, \ldots, w_n\}$ we create hash by arbitrary hash function H (e.g. MD5 128b) $H(D) = \{H(w_1), H(w_2), \ldots, H(w_n)\}$ so we have every single word represented by 128 bit hash. Then we iterate over all particular bits through the words from given list and then set 1 if there is more words which has 1 at the particular position, or set 0, otherwise. Figure 1.4 shows intuition about creation of Simhash for three different documents with 7 words. Dark square denotes one-bit, white square denotes zero-bit.

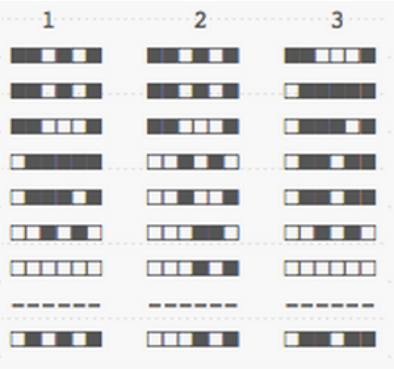


Figure 1.4: Principle of the Simhash creation

Chapter 2

Analysis and design

This chapter describes the complete analysis and design of a system for the creation of new text signals for full-text relevance. First of all we will provide description of available datasets (2.1), second we introduce our feature quality evaluation system (2.2), from the very beginning phase of creating features through evaluation to the final stage printing results. Third we will describe created text signals (features) (2.3) that were evaluated afterwards. Finally we will describe design of our query expansion system (2.4).

2.1 Datasets

Datasets for information retrieval, as well as for other machine learning tasks, are used for training and evaluation machine learning model. In this case we use private datasets provided by Seznam.cz to find and evaluate new text signals and publicly available dataset LETOR [24] to compare own implementation of the AdaRank with other LTR algorithms.

2.1.1 Seznam Datasets

We have divided Seznam datasets to two different category. First one is called feature dataset that contains query–document lists with baseline signals used by Seznam.cz. Second dataset is more text buffer than classic dataset and it contains just text information about the documents from feature dataset.

2.1.1.1 Feature dataset

Format of the feature dataset is displayed in a well arranged way on a Figure 2.1. Every single row in the dataset represents query–document pair which has assigned relevancy label and feature vector. Specifically every row r respects standard structure of SVM light [25] format:

```
r := <relevance label> qid:<qid_value>
```

```
<key_1>:<val_1> ... <key_n>:<val_n> # <query_tokens>
<doc_url> <doc_hash> <doc_age> <query_doc_id>
```

Dataset provided by Seznam.cz contains circa 45 thousands of queries and more than 2 millions of query-pair documents. That means that average number of documents per query is equal to 33.45 which corresponds to the other different public dataset.

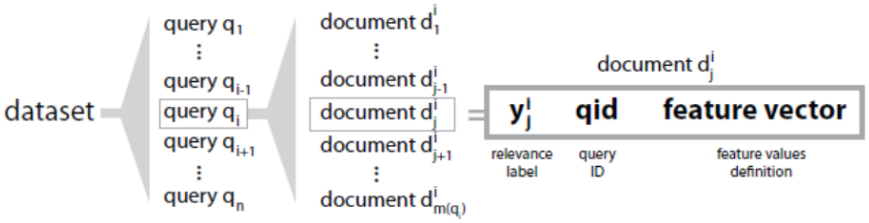


Figure 2.1: SVM light format

Every query–document pair contains list of anonymised feature set that serve just as the baseline for comparison with the new ones. One reason why we have only anonymised feature set is obviously for the security purposes. Another reason is that Seznam.cz did not want to have any influence on our own research.

Table 2.1: Baseline signals provided by Seznam.cz

Description	Number of features	ID ranges
Query signals	55	100 - 163; 894 - 924
Main link signals	9	600 - 608
Document signals	41	609 - 649
Feedback signals	14	758 - 771
Text signals	**11**	773 - 783
Mixed signals	25	733 - 757
Other signals	249	650 - 732; 784 - 893; 925 - 979
All	404	100 - 979
Used	130	100 - 649; 758 - 783; 894 - 924

Nevertheless Seznam.cz ensures description for feature ranges so we do know approximately type of each signal with given ID. The Table 2.1 introduces these types and its basic statistics. We can see that there were total of 404 signals, however we use just 130 - category "Other signals" contains experimental and redundant signals which were not recommended to use by Seznam.cz.

The most important section are the text signals since the main goal of this work is to improve full-text relevance by the text signals so we want compare our new signals especially with these. The values of ID ranges are important for experimental results in Chapter 4, as it serves to identify the individual baseline signals.

Query statistics We have performed some computations over the dataset and focused also on the queries. Following graph 2.2 shows query length distribution at the dataset so we can see that queries are relatively small. Average query length is equal to 2.41 and there is more than 10 thousands queries with just one token.

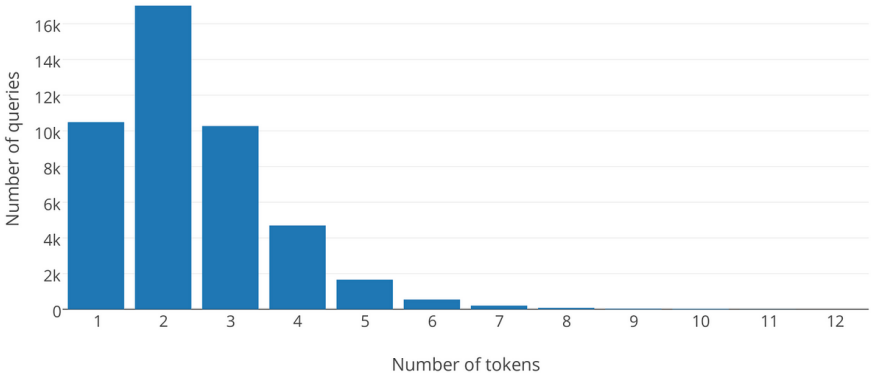


Figure 2.2: Query length distribution

Feature dataset proprocessing Within preprocessing of the feature dataset we have removed experimental and duplicated signals from the dataset as mentioned above so we worked with 130 different baseline signals provided by Seznam. Complete preprocessing process is showed at Figure 2.3. The stage called "Preprocessing" includes the signal selection and normalization to zero mean and unit variance. Subsequently there were created three baseline feature dataset; first one with all used signals, second one with just eleven text signals and last one with all non-textual signals - that means complement of the text dataset.

2.1.1.2 Content dataset

Content dataset contains metadata and text data from the web document with total size of 600 GB and millions of documents. Structure of the dataset is in Google Protocol Buffer [26] format which provides fast read even very huge data. The dataset includes connections between web document via inlinks and outlinks in addition to the texts from document content.

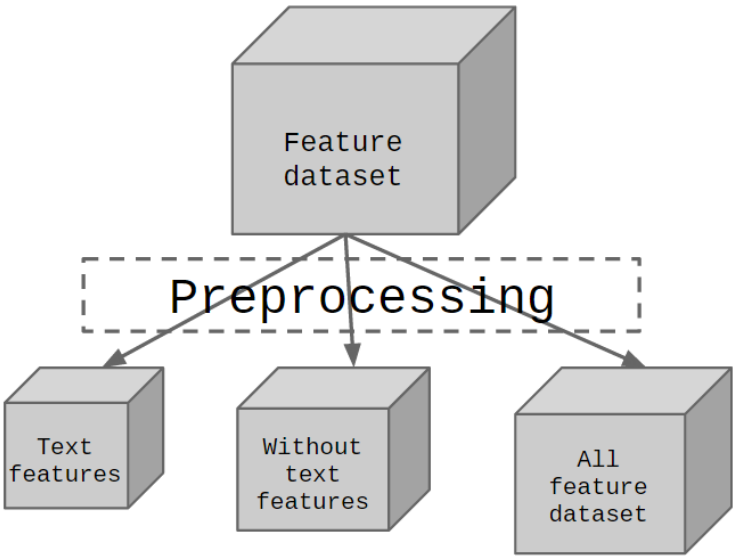


Figure 2.3: Schema of feature dataset preprocessing

Nevertheless for the purposes of this work we have focused only on these (meta)data:

- textual content from the document
- average word length for given document paragraph
- average anchor text length for given document paragraph
- number of divs before given paragraph
- number of tags inside a paragraph
- number of tags before before a paragraph
- document size

Content dataset preprocessing Within content dataset preprocessing was needed to select only the documents for which we had baseline signals at the feature dataset. Pre-selection phase was particularly important because of computation time and memory requirements.

On these pre-selected documents we have gathered information from document content, title, headers (that means text from H tags), url and combination all together. These data are really important because it serves to compute a lot of new signals.

Moreover for every texts were applied sets of text preprocessing methods. This means that for every above-mention section (i.e. content, title, ...) from document were applied stemming (two-phase) and lemmatization, which led to the fact that several text sources were created. We prepared 20 different textual sources to create new signals from the document (variation of 4 preprocessing methods and 5 document sections).

2.1.2 LETOR dataset

LETOR [24] is a package of benchmark data sets for research on LEarning TO Rank, which contains standard features, relevance judgments, data partitioning, evaluation tools, and several baselines.

Specifically, LETOR contains two main datasets. The first one LETOR3.0 was released in 2008 and it uses Gov web page collections and older OHSUMED data collections. While second one LETOR4.0 is totally new release. The datasets respects the same format as the dataset from Seznam.cz therefore it is really easy to use both with the same process schema.

Unlike the Seznam datasets, we use LETOR to compare own implementation AdaRank LTR algorithms with others and do not use to create new features. This is because we do not have content dataset for this query - documents.

2.2 Feature datasets evaluation

Feature datasets evaluation is designed that it is easy to test and add new text signals to datasets. Figure 2.4 visualize all concept and schema of the evaluation system and following paragraphs describe particular stages deeply.

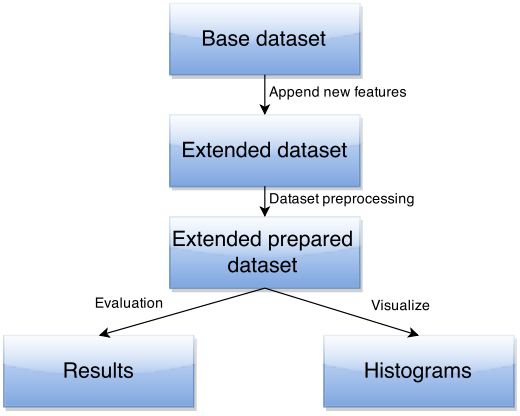


Figure 2.4: Feature quality evaluation schema

Append new features is a phase in which a set of given new features is appended to the base dataset. The dataset can be every dataset which respects SVM light format. In our case the base dataset denotes one of the aforementioned preprocessed feature dataset provided by Seznam.cz. Due to the structure of the SVM light format it is really easy to provide it and the only one requirements for a new feature is to respect the same SVM format (which is defined as follows pair separated by dash `<key>:<val>`).

Dataset preprocessing covers classical normalization to zero mean and unit variance and also dataset division to train, validation and test dataset in the ratio of 60:20:20. Moreover there is a dataset file conversion to binary format for faster evaluation in the following system stages.

Evaluation is the most interesting and the most important part of this feature evaluation system. At this stage there is carried training set of Learning to rank algorithms, specifically was designed follows:

- LambdaMART

  According to the [4], it is state-of-the-art LTR algorithm

- RcRank

  Algorithm which is used by Seznam.cz.

- AdaRank

  Own modificated implementation.

Every LTR algorithm is design to run in parallel and use all three (train, valid, test) dataset in standard machine learning manner. After the train these model there is evaluation phase that includes following evaluation measure:

- NDCG@10

  Representant of the Position model evaluation measure.

- ERR@10

  Representant of the Cascade model evaluation measure.

- SR@20

  Evaluation measure currently used in Seznam.

Visualization phase serves just to show what kind of distribution the new signals have.

2.2.1 Signal importance

Thanks to the aforementioned evaluation model we are able to determine if the new dataset is better or worse than baseline. Nevertheless we would not be able to determine which new signal from the dataset is the most important and has the most discriminative power.

This task is typically treated by decision trees. The relative rank (i.e. depth) of a feature used as a decision node in a tree can be used to assess the relative importance of that feature with respect to the predictability of the target variable.

Let us consider following assumption that features used at the top of the tree are used contribute to the final prediction decision of a larger fraction of the input samples. The expected fraction of the samples they contribute to can thus be used as an estimate of the relative importance of the features.

Hovewer this approach using decision trees has meaningful drawbacks. Let there are two signals s_1 and s_2. It may indeed happen that signal s_1 never occurs in any split because it leads to splits that are slightly worse, and therefore not selected, than those of some other variable s_2. Nevertheless, if we remove s_2 and construct new tree, s_1 may now occur prominently within the splits and the resulting tree may be almost as good as the original tree with signal s_2. This is called **masking effect** [27].

Random forest are able to overcome the problem of masking effect by its principle. The algorithm is briefly described in Chapter 1, nevertheless the key point is that during the construction of the decision tree, in every split, the set of signals is subsampled in a random manner. Thanks to randomization, masking effects are reduced within forests of randomized trees. Even if there are two very similar signals s_1 and s_2 as in the case above, there is still a chance for slightly worse signal s_1 to be chosen as a split if s_2 is not selected within the signal subsampling.

2.3 Used signals

This section describes designed signals we test and compare in our implemented evaluation system. We divide the signals to three main category. First category is called handcrafted features and covers features mainly inspired by meta-data from content dataset and also by publicly available Google text signals for information retrieval. Second one is based on Vector space model, which was transformed to different spaces, especially TF-IDF, LSI and LDA. The signals are then created as a cosine distance between query vector and document vector. The last category covers "semantic" features which was experimentally designed and based on Deep learning and Simhash algorithms.

Name	Type
Average length of anchor texts through all paragraphs	Document
Average number of DIVS in front of every paragraph	Document
Document length	Document
Average number of tags through all paragraphs	Document
Average number of words through all paragraph	Document
Average length of the words	Document
Average number of tags in front of every paragraph	Document
Query token is the most frequent word from document	Query / Document
Query token is the most frequent word from document content	Query / Document
Query token is the most frequent word from document title	Query / Document
Query token is the most frequent word from document URL	Query / Document
Query token is the most frequent word from document headers	Query / Document
Query token is in TOP 5 frequent word from document	Query / Document
Query token is in TOP 5 frequent word from document content	Query / Document
Query token is in TOP 5 frequent word from document title	Query / Document
Query token is in TOP 5 frequent word from document headers	Query / Document
Query token is in TOP 5 frequent word from document URL	Query / Document
Query token is in TOP 10 frequent word from document	Query / Document
Query token is in TOP 10 frequent word from document content	Query / Document
Query token is in TOP 10 frequent word from document title	Query / Document
Query token is in TOP 10 frequent word from document headers	Query / Document
Query token is in TOP 10 frequent word from document URL	Query / Document
Number of query tokens in document	Query / Document
Number of query tokens in document content	Query / Document
Number of query tokens in document title	Query / Document
Number of query tokens in document headers	Query / Document
Number of query tokens in document URL	Query / Document
Inverted best match token in document	Query / Document
Inverted best match token in document content	Query / Document
Inverted best match token in document title	Query / Document
Inverted best match token in document headers	Query / Document
Inverted best match token in document URL	Query / Document
Query token density in document	Query / Document
Query token density in document content	Query / Document
Query token density in document title	Query / Document
Query token density in document headers	Query / Document
Query token density in document URL	Query / Document

Table 2.2: Handcrafted signals

2.3.1 Handcrafted signals

We provide list of all handcrafted signals at Table 2.2. The first seven handcrafted signals in the table are dependent only on the document metadata that means it does not matter on particular query. These signals are directly taken from metadata content dataset. Nevertheless, rest of handcrafted signals is crafted from lemmatized document content itself and it is dependent on query and on document as well.

All these signals are self-explanatory except Query Token Density and Inverted Best Match Token in Document that require at least a brief description. The first mentioned signals is computed as follows. Let c_d is a document word count and q_d is a number of the words from document that are occurred in query as well at the same time. Query token density QTD is then equal to:

$$QTD = \frac{q_d}{c_d} \tag{2.1}$$

Inverted Best Match Token $IBMT$ is computed for given query $q = \{q_1, \ldots, q_n\}$ with n query tokens and document $d = \{d_1, \ldots, d_m\}$ with m document tokens sorted in descending order by its frequency as follows:

$$IBMT = \frac{1}{min_{m_1,\ldots,m_m} d_m \in q} \tag{2.2}$$

2.3.2 Vector space model features

The entire process of generating signals is quite complicated so the best way how to introduce this process it is its visualization to the scheme. The scheme (see Figure 2.5) covers six different stages from crafted document text from a content dataset to creation of new signal.

First there is **text processing** which covers such a preprocessing method as tokenization (provided by Seznam), lower casing, lemmatization and stemming. From the preprocess text we **create dictionary** from different positions of document (content, URL, title, headers, all together) that it is pruned to different sizes. Main reason why we prune these dictionaries is that some algorithms and transformations need to have input with lower dimension (e.g. neural networks or LSI/LDA). Pruning also serves as a removing stop words thanks to filtration of over-frequent words. Subsequently we **create corpus** via each document parts and dictionary size. The corpus has bag-of-words structure that means that for each document we have list of word ID (crafted from dictionary) and its frequency.

Corpus transformation covers transformation typically from bag-of-words model to more complex vector space models. List of transformations is follows:

- TF-IDF

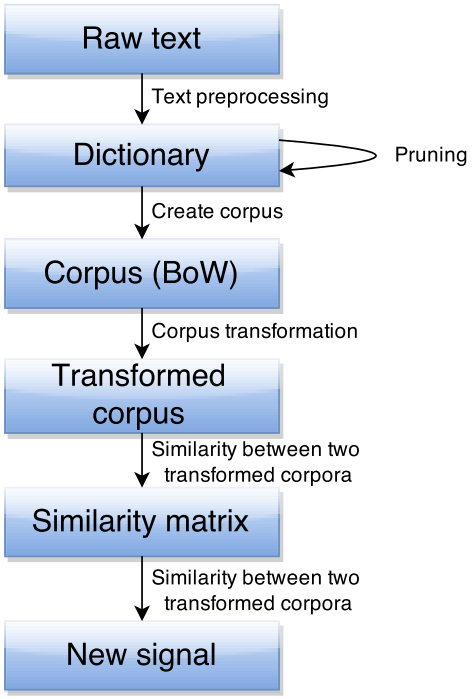


Figure 2.5: Scheme to create new VSM signals

- LSI

  We perform different transformation with different number of topics.

- LDA

  We perform different transformation with different number of topics.

After the transformation we are able to compare these document vectors with given query vector for instance in cosine similarity measure therefore we create so called **similarity matrix** representing distance between query and document. We assume that closer vectors are more similar.

2.3.3 "Semantic" signals

Group of these signals covers both Semantic hashing and Simhash. On the one hand VSM signals and "Semantic" signals overlap each other because both groups are kind of semantic signals and try to find some latent space of document and queries. Nevertheless Semantic hashing and Simhash requires different preprocessing and treatment in general, so we decided to create special category.

2.3.3.1 Semantic hashing

It is designed and proposed several variants of the signals produced by Semantic hashing. The main problem that needed to be resolved is a very high dimension of query / document vectors. It is impossible to learn such a high-dimensional input layer of deep neural networks. The way to tackle this problem is to reduce input dimension on the acceptable value in thousands.

First option how to reduce the dimension of document / query vectors is to prune its dictionary. Original paper, which introduces Semantic hashing, describe dictionary pruning to TOP 2000 the most frequent word in documents. This variant has a major drawback that we loose a large part of the information.

Another approach how to reduce vector space dimensionality is so called Word Hashing [28] and so called "average Word2Vec" [29] which we describe in following paragraphs.

Word Hashing is an efficient way how to reduce the dimensionality of the bag-of-words term vectors. It is based on letter n-gram. Given a word (e.g. wood), we have to first add starting and ending marks to the word (i.e. #wood#) and consequently we break the word into letter n-gram (e.g. letter 3-grams: #wo, woo, ood, od#). Finally, the word is represented using a vector of letter n-grams.

There is obviously problem with collision; that means two different words have the same vector of letter n-grams. Nevertheless the collisions are pretty rare and according to [28] there is negligible collision rate of 0.0044% for sample dictionary with dictionary size equal to 500 thousands and letter n-grams with $n = 3$. Moreover they were able to reduce the dictionary to dimension equal to 30 thousands which means almost 16 times lower dimension.

Average Word2Vec This approach to reduce dictionary dimensionality is based on Word2Vec algorithm. Word2Vec was developed by Mikolov as his dissertation [30] in 2013. Mikolov found out that synaptic weights between input and hidden unit in the neural net, which is trying to predict following word for given sequence of words, represent n-dimensional vector with interesting properties. For our task the most important one it is that if there are two words with relatively similar meaning there is relatively small distance in between as well. Another interesting property is that we can specify dimension size of word2vec algorithm in advance.

Therefore we can interpret each word in document with arbitrarily large dimensions. The paper [29] shows that we can interpret even whole document just by averaging all words word2vec representation from document with reasonable results.

Finally we get low-dimensional representation for each document in corpus, moreover with predefined size n.

2.3.3.2 Simhash

Principle of Simhash signals is simply based on Simhash algorithm. We preprocess both the documents and queries and then for each query and document create appropriate simhash. Due to the use MD5 hash algorithm we represent all documents and queries by 128 dimensional space and therefore the resulting simhash has dimension equal to 128 as well. Finally we compare simhash of document and query by cosine similarity measure which is used as our feature.

2.4 Query expansion

In the context of web search engines, query expansion involves evaluating a user's input (what words were typed into the search query area, and sometimes other types of data) and expanding the search query to match additional documents. Query expansion involves techniques such as (I) finding synonyms of words, (II) finding all the various morphological forms of words by stemming, (III) fixing spelling errors and automatically searching for the corrected form or suggesting it in the results and (IV) re-weighting the terms in the original query.

At this work, we focus only on finding synonyms part. Considering the fact that average query has 2.4 query tokens, it is reasonable idea to somehow expand these queries to make the query vector more dense. There is an assumption that if we will be able to represent each query with multiple tokens we would be able to more precisely predict its similarity with respect to a given set of documents.

As already mentioned in the paragraph about Word2Vec, the vector representation is able to find semantically similar words to arbitrarily given word. We try to train the Word2Vec model on our document corpus and then for each token from query find TOP5 the closest ones. Our hypothesis is that the set of closest vectors (tokens) should be semantically the most similar words which corresponds with finding synonyms.

Chapter 3

Realization

This chapter provides a detailed description of the means that were used during implementation. First we briefly describe used computational external resources, second we provide information about used programming languages and external libraries and finally we provide insight into the implementation details of our work.

3.1 Used computational sources

Due to the fact that we had available data on the order of gigabytes, it was more than practical to use some external computation sources. For the purpose of this work we use MetaCentrum [31] resources. MetaCentrum Virtual Organization operates and manages distributed computing infrastructure consisting of computing and storage resources owned by CESNET as well as a lot of co-operative academic centers within the Czech Republic.

3.2 Used programming languages

3.2.1 Python

For implementation was used programming language Python which is commonly used in the data mining algorithm field. Furthermore, Python provides a lot of libraries for very easy and efficient computations over the matrices, especially Numpy, that provides comparable time efficiency with the pure implementation in C language. Moreover it is ideal programming language for prototyping and experimenting.

3.2.2 Bash

We use interpreted programming language Bash to create jobs on MetaCentrum. Jobs are relatively small scripts that work as controllers. Common

structure of these jobs is follows:

1. it allocates appropriate computational resources (CPUs, RAM, HDD) to a given python program;

2. it copies input files from storage to assigned computational node;

3. it runs appropriate program itself;

4. it copies output files from the computational node to the user storage

3.3 Used external libraries

Scikit-learn [32] is an open source machine learning library for the Python. It provides various classification, regression and clustering algorithms and its use is extremely straightforward. We use this library to compute and visualize signal importances, its implementation of the regression trees used in our AdaRank and many other utils functions.

RankPy [33] provides efficient implementation of the LambdaMART algorithm, which is the state-of-the-art LTR algorithm. We use this library also to read traditional SVM-light dataset format and convert it to query - document objects.

RankLib [34] is a library of learning to rank algorithms implemented in Java. Currently several popular algorithms have been implemented, we use implementation of RankSVM, ListNet and classic version of AdaRank.

Gensim [35] is very useful open-source tool for topic modeling, text transformation and computing document vs query similarity developed by Řehůřek as part of his PhD thesis. Gensim includes implementation of TF-IDF, LSI, LDA, deep learning with Google's word2vec and much more algorithms using in natural language processing.

Theanets [36] package provides tools for defining and optimizing several common types of neural network models. It uses Python for rapid development, and under the hood Theano provides graph optimization and fast computations on the GPU. Usage is similarly straightforward as in case of Scikit-learn library.

3.4 Implementation

During the whole implementation we used Gitlab repository which is based on GIT and provided by our university. It served as a distributed revision control system and private backup storage as well. We were able to share our code in different places, for example in different local computers, MetaCentrum storage discs, etc. which was a huge advantage of this system.

The biggest part of our implementation is dataset preprocessing itself, which covers several separate scripts. Most scrips is composed of two files. First is script itself and covers script functionality, second one is so called "run script" containing CLI [4] for interaction with the external environments and launch the script.

3.4.1 Feature datasets evaluation system

The purpose of this system is to measure dataset quality and compare it with other datasets. Whole system is represented by benches of bash scripts which are configured to run on MetaCentrum with arbitrary parameters. To run example script `test_new_signal` we can use for instance these command:

```
qsub -v BASE=base_text_features,STANDARDIZE="-s",
SIGNALS_DIR=handcrafted,TYPE=all test_new_signal.sh
```

Qsub command executes a new job on the MetaCentrum and parameter `v` defines the input variables for the script. `BASE` denotes base dataset, `STANDARDIZE` sets whether we want to standardize the new tested signal, `SIGNALS_DIR` defines directory containing the tested signals in format `<key>:<val>`, and finally `TYPE` determines in which mode we want to run the script.

The script itself has three different modes:

- **prepare**

  Due to the fact that the dataset evaluation by individual LTR algorithms takes even tens of hours, there is a possibility to run only part of the process. This mode creates a new extended dataset with new signals, normalize signals, transform dataset to binary form and finally computes signal importance by the Random Forests.

- **eval**

  This mode evaluate the extended dataset on set of LTR algorithms. We assume that we ran Prepare mode before, so all prerequisites are already created.

[4] Command line interface

- **all**

  This mode combines both previous. It internally runs prepare mode first and then runs eval mode.

3.4.1.1 Evaluation measures

Within the implementation of our evaluation system we use the NDCG evaluation measure provided by Rankpy library, nevertheless we had to implement own ERR and SR measure. Our implemented version of these measures were then integrated into the library as well for better usage. Moreover, we obtain secret parameters from Seznam to adjust SR measure which is not publicly available.

3.4.1.2 Provided results

Each test of the new signals provides several result files for each LTR algorithm separately. First type of these files is so called "configures" files that contains all parameters defined for particular algorithm in particular test. Second there is saved the LTR model itself to re-use the same model for different signal test. Third we provide output file for each evaluation test that contains particular training phases, training and evaluation errors for each round and the test error at the end of this output file. Finally there is signal importance files that is described in following paragraph.

3.4.1.3 Signal importance

Signal importance is implemented as a python run script with CLI using RankPy library to import input dataset with given tested signals to standard object query / document form and regression random forests, implemented in scikit-learn library, to compute signal importance. Within the script there are two different outputs. First, output in the form of a tabular listing and secondly graphic output in graph form.

3.4.2 Semantic hashing

Implementation of Semantic hashing is divided to two stages. First stage is preprocessing which are implemented word hashing and average word2vec methods. Second one covers implementation of Semantic hashing itself.

3.4.2.1 Preprocessing

To preprocess high dimension corpus to the lower one we implement two types of preprocessing. Within these preprocessing we use lemmatizated corpora. Implementation of average word2vec and word hashing is located at the same scripts. We use Gensim word2vec implementation which is optimized

in `Cython` [5] and parallelized for faster execution. There is also easy scalability and several adjustable parameters such as number of iteration, size of word2vec vectors and minimal word count which reduces the words that are almost absent in the training corpus.

3.4.2.2 Algorithm

To implement semantic hashing, especially to construct stacked denoising autoencoders, we use theanets library. It simply allows to adjust several parameters such as the size and structure of the neural network, the possibility of adding noises, etc.

Our script has several adjustable parameters:

- **number of layers** (int)

  This parameter represents the depth of the created network. The depth is measured from the input layer to the middle layer.

- **middle layer** (int)

  The size of the middle layer, this parameter practically features the size of latent output representation.

- **hidden noise** (float)

  The level of noise added to the hidden layer of the created network.

- **input noise** (float)

  The level of noise added to the input layer of the created network.

- **batches**

  The batch size to accelerate the training phase.

- **optimize**

  Inner parameter of theanets library. It defines what type of neural network learning will be chosen. Layerwise is a default option, that means the neural nets is learned layer by layer by `RmsProp` sequentially.

  The RmsProp method uses the same general strategy as Stochastic Gradient Descent (SGD), in the sense that all gradient-based methods make small parameter adjustments using local derivative information. The difference here is that as gradients are computed during each parameter update, an exponential moving average of gradient magnitudes is maintained as well.

[5] Cython is a compiled language that generates CPython extension modules. These extension modules can then be loaded and used by regular Python code using the import statement. It is actually a Python to C source code translator that integrates with the CPython interpreter on a low level.

Due to the fact that theanets library handle with dense numpy matrix it is impossible to train all data at the same time. We have implemented "batch" training which divides the input data into smaller chunks and provides sequentially several training phases that means the neural network is continuously updated.

3.4.3 Simhash

We implemented own Simhash algorithmus according to [37]. Unless the original Simhash which uses just TF weighting for each word in document, we use TF-IDF weighting implemented by Gensim library. To hash every single word to MD5 128b structure we employ standard python `hashlib` library that provides several hashing functions.

3.4.4 Query expansion

Implementation of query expansion use Gensim's word2vec algorithm and try to find the TOP 5 closest word to a given token from query. Within text preprocessing the query tokens are lemmatized. The script is parallelized by dividing the input file into several smaller parts - chunks. Each chunk is placed into a queue to be processed sequentially by individual threads. This functionality is covered by `joblib` python library.

Output of this script is a feature dataset, which is extended by the word2vec "synonyms".

3.4.5 AdaRank

We have implemented own modified version of AdaRank algorithm which we prepared for the extension of the RankPy library.

Principle of the algorithm is already described in the research chapter, nevertheless unless the classic implementation of AdaRank algorithm where is used just basic linear weak ranker, this implementation introduces shallow regression trees as the weak ranker. We use regression tree implemented in the scikit-learn library.

Moreover, within this work were implemented two approaches of updating data distributions P. The first one is updated by dataset re-sampling according to a weighted distribution P_{t+1} every boosting round t. The second implementation weights the dataset over the queries more straightforwardly through the parameter `weight samples` from Scikit-learn library during a learning the weak ranker each round t.

Chapter 4

Experiments

In this chapter we show our experimental results. First we compare our implementation of AdaRank algorithm with the other LTR algorithms. Second we provide evaluation results of designed signals - how good the features are in comparison to baseline signals provided by Seznam.cz.

4.1 AdaRank experiments

It has been conducted several experiments to compare our implemented AdaRank performance with another learning to rank algorithms. One of the compared algorithm is state-of-the-art – according to Yahoo! Learning To Rank challenge, LambdaMART, that combines regression boosting tree and neural network approaches. Most of the best results were achieved by using LambdaMART algorithm in this competition. Furthermore, these experiments present AdaRank performance results in compare to different tuning parameters which are based on our implemented regression trees as a weak ranker. Table 4.1 shows all the parameters and its testing values.

Table 4.1: Tuning parameter set for the MQ2007 dataset

parameters	values
max depth	1, 2, 3, 4, 5, 6, 7
min samples split	2, 4, 6, 8, 10, 12, 14, 16
max leaf nodes	None, 2, 4, 6, 8, 10, 12, 14, 16
NDCG cut off	1, 5, 10

4.1.1 Experiment Settings

As already mentioned, LambdaMART was selected as baseline in the experiments, because it is the state-of-the-art learning to rank method. Further-

more, for the lowest bound of ranking documents was chosen so-called 'RandomRank' which represents basically random documents shuffling. Thanks to this algorithm we can see how many times is our implementation better than random ranking of documents.

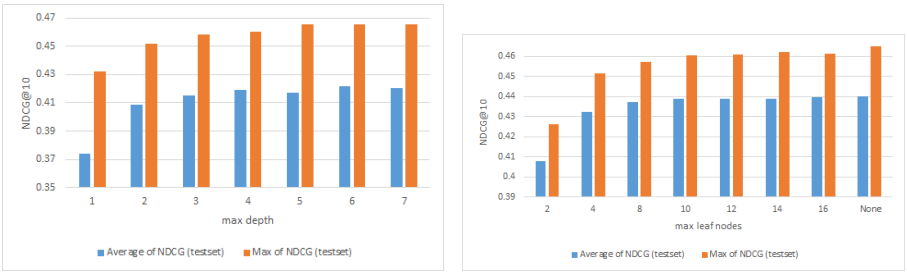
For AdaRank, the parameter T was determined automatically during each experiment. Specifically, when there was no improvement in ranking accuracy in terms of the performance measure during a x rounds, the iteration stops (and T has been determined). The parameter x can be chosen dynamically by one of the input parameter. Another parameters (see Table 4.1) for Regression tree weak rankers have been manually chosen depending on the dataset.

4.1.2 Experiments with the MQ2007 dataset

In this experiment, the dataset MQ2007 has been used. MQ2007 is only one part of the LETOR4.0 datasets [24] that is provided by Microsoft research group. The MQ2007 consists of 1700 queries and 70k documents (approximately 40 documents per query), but the dataset is divided to the 5-fold cross validation partitions. Each partition consists of training dataset, validation dataset and test dataset. These experiments were conducted for all 5 folds and compared afterwards.

Every row in the dataset represents a query-document pair and its structure is classical `SVMlib` format. The first column is relevance label of this pair, the second column is query id, the following columns are features, and the end of the row is comment about the pair, including id of the document. The larger the relevance label, the more relevant the query-document pair is. A query-document pair is represented by a 46-dimensional feature vector.

First part of the experiments is focused on the AdaRank tuning parameters, such as `maximal depth` of regression tree, `minimal count of samples` to split tree nodes, maximal number of leaf nodes created during the training phase and the last one is `cut off` in evaluation measure, in this case NDCG measure.



(a) Maximal depth (b) Maximal leaf nodes

Figure 4.1: AdaRank - parameters - NDCG evaluation measure

The results reported in Figure 4.1a shows meaningful dependency between

accuracy and max depth parameter. More complex tree (our weak ranker) through deeper structure of tree means better results. We can see the biggest gap between depth equals to 1 and depth equals to 2, that is relatively evident because a tree with depth equals to 1 provides only linear separability.

Figure 4.1b shows very similar correlation, in this case complexity of tree is represented by maximal leaf nodes parameter, now it is evident that tree with more leaf nodes is more complex.

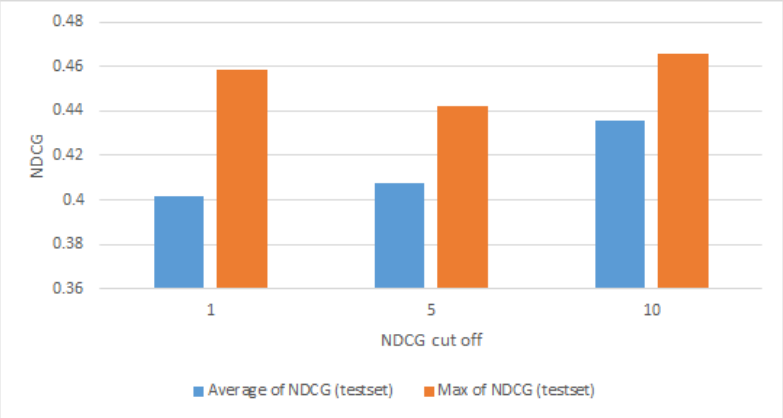


Figure 4.2: NDCG on the test set for different values of NDCG cut off parameter

Another experiments prove that the smaller cut off parameter (cut off means we are interested just in the first k items from ranked list) provides worse results. The experiments result is given in Figure 4.2.

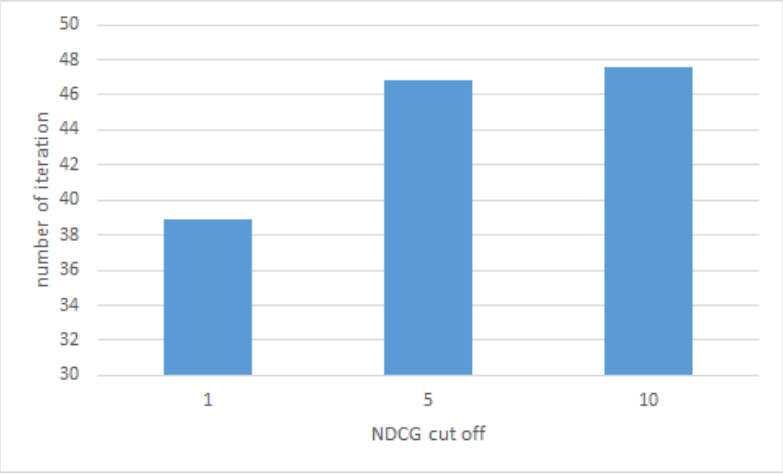


Figure 4.3: Number of iterations depending on the NDCG cut off parameter

Very interesting results are given in Figure 4.3 and 4.4. We can see that number of iteration is quite small in compare to results in AdaRank which has been implemented by Jun Xu and Hang Li [11]. This implementation converged up to 60 iteration independently on the NDCG cut offs. In the Figure 4.3 we can see slight growth for smaller cut off, but it is not considerable gap. In the implementation of Jun Xu and Hang Li is presented learning curve with over than 300 iterations, that means growth over than 4 times more. It could be consequence of using different weak ranker since they use really trivial one.

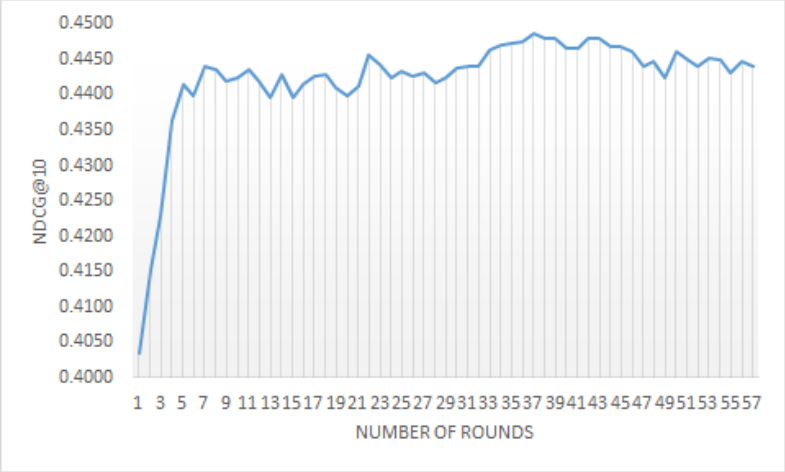


Figure 4.4: Learning curve of AdaRank algorithm

Experiment results which are showing the AdaRank results for different folds of the MQ2007 dataset are given in Figure 4.5. For this results there was chosen AdaRank configuration according the previous experiments which gives the best results on this dataset. We can see that results for cut off NDCG equals to 1 gives unexpectedly high results. It could happen as a result of the unstable nature of NDCG@1. AdaRank algorithm consists a lot of random effects and during NDCG@1 is really important every change in ranking set - because we are interested only in the value of first document. Nevertheless, the results show that there is practically no difference between NDCG@1 and NDCG@5 in the cross validation fold of dataset. Furthermore, the results for NDCG@10 are far higher than the others.

The last experiment compares a few learning to rank algorithm with AdaRank algorithm on the cross validate fold of the MQ2007 dataset. Results of an algorithm RandomRank (represents really naive approach) is provided only for comparison with real (non trivial) rankers. We can see that results of particular rankers are almost the same for this dataset.

We investigate the dependencies between particular parameters of weak

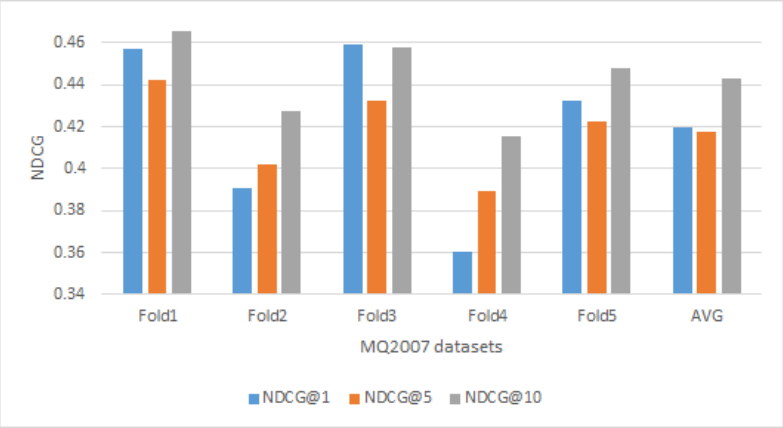


Figure 4.5: AdaRank NDCG results for different folds of MQ2007 dataset

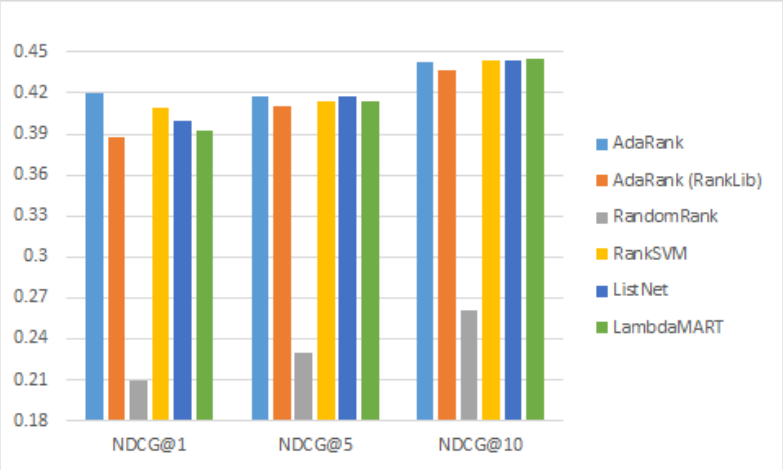


Figure 4.6: Comparison of ranking accuracies for the MQ2007 dataset

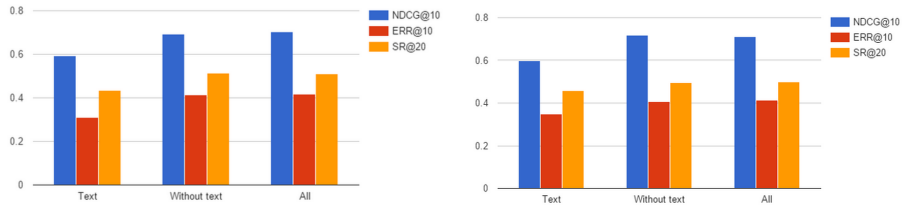
ranker and NDCG measures and prove that more complex regression tree as a weak ranker of AdaRank algorithm provides two main benefits compare to trivial weak ranker. The first one is the higher speed of convergence (a smaller number of iterations) and the second one is the better results measured in NDCG evaluation measure. Nevertheless it is expected that the same results will be given for another evaluation measure, specifically for the MAP or ERR measure.

We also show (see Figure 4.6) that choice of the learning to rank algorithm is not the key aspect for ranking documents. Much more important seems to be the feature selection from the documents and datasets itself. The following section will pursue the testing of analyzed and designed new text signals.

4.2 Baseline signal datasets experiments

As already mentioned in chapter 2, we divided the feature dataset with baseline signals provided by Seznam.cz into the three subdatasets. First dataset contains just text signals, second one only non-text signals and the last one includes all baseline signals.

In this experiment we compare the quality of those subdatasets via several evaluation measures, especially NDCG@10, ERR@10 and SR@20, and via several LTR algorithms (RcRank and LambdaMART).



(a) RcRank (b) LambdaMART

Figure 4.7: Signal dataset comparisons

In the figures 4.7a and 4.7b we can see there is just a negligible quality gap between non-text signal dataset and all signal dataset in the all of tested evaluation metrics. That means that text baseline signals do not bring any improvement to relevance ranking. Moreover we can observe significantly worse evaluation results for text signals itself in compare to non-text or all signals.

4.2.1 Baseline signal importance

The all baseline signals are anonymized therefore we do not know the exact meaning of particular signals. Nevertheless as we already provided in Table 2.1 we know at least approximate signal type for given ID ranges.

Signal type	Number of signals
Query signals	2
Main link signals	1
Document signals	2
Feedback signals	8
Text signals	7

Table 4.2: Representation of groups in TOP 20 the most important signals

Following result tables and graphs will contain only the ID of the individual baseline signals. First graph in 4.8 shows relative importances across the all 130 baseline signals. We can see that there are three really important

signals, however, at least about half of the signals has considerably high importances as well. Table 4.2 provides frequency representation for each signal groups. Surprisingly we can see that seven of them are text signals nevertheless according to signal dataset comparisons it do not bring any improvements by any evaluation metrics. That means text signals are compensated with different signals, the most likely with feedback signals, such as signals crafted from click stream and user behavior.

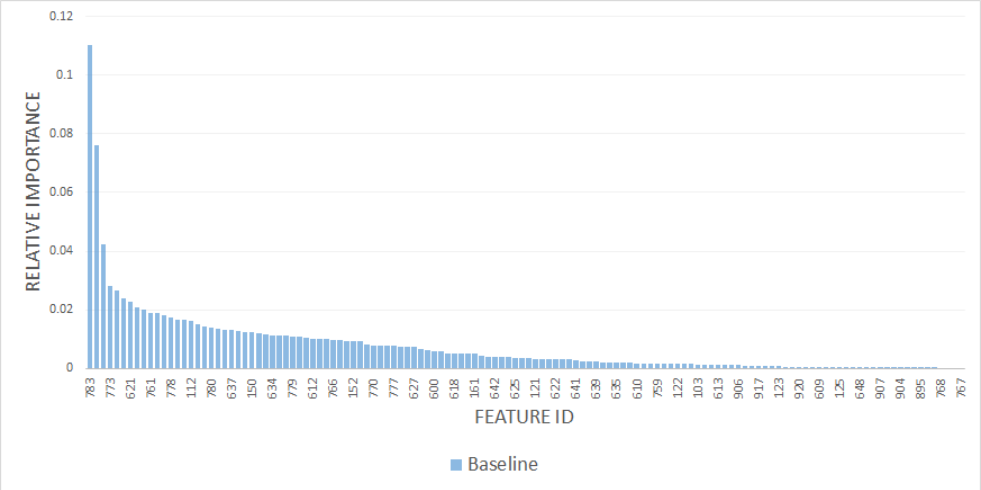


Figure 4.8: Baseline signal relative importances

4.3 Global settings for signal experiments

Since next sections will provide results of experiments over the text signals we mention general settings for these experiments.

4.3.1 LTR configurations

To evaluate text signals we use three different LTR algorithms with the following configuration:

- LambdaMART

  These settings are recommended by rankpy library for its implementation of the LambdaMART algorithm.

  - estimators = 10000

    The number of regression tree estimators that will compose this ensemble model.

  - estopping = 100
    The number of subsequent iterations after which the training is stopped early if no improvement is observed on the validation queries.
  - shrinkage = 0.08
    The learning rate (also known as shrinkage factor) that will be used to regularize the predictors (prevent them from making the full (optimal) Newton step.
  - max depth = 4
    The maximum depth of the regression trees.

- AdaRank

  AdaRank settings have been adjusted according to previous experiments in section 4.1.

  - iteration = 100
    The number of boosted rounds (iterations).
  - max leaf nodes = Inf
    The maximum leaf nodes in the regression trees.
  - max depth = 7
    The maximum depth of the regression trees.
  - min samples split = 2
    The minimal number of samples to split tree node.

- RcRank

  To perform evaluation experiments with Seznam's RcRank we gave the binary executable file with the algorithm and recommended settings which provide optimal results for given evaluation tasks. For reasons of security we do not post these parameters publicly.

4.3.2 Evaluation configurations

For experiments with the text signals we use three different evaluation measures which evaluate all the above mentioned algorithms LTR algorithms.

1. NDCG@10

   Representative of the positional model evaluation measures with cut-off equal to 10.

2. ERR@10

   Representative of the cascade model evaluation measures with cut-off equal to 10.

3. SR@20

   Evaluation measure implemented and used by Seznam.cz. They provide appropriate parameters.

4.4 General structure of the experiments

Since we have conducted several experiments over the all of aforementioned text signals in Chapter 2 and each of them have very similar structure, in this section we describe the general structure for most following experiments.

First before every experimental results we provide list of tested signals with assigned IDs. Each extended signal is numbered from 1000 so it is easy to determine whether the signal ID is within the baseline or it is extended one.

Second there are showed signal importance results performed by regression random forest to compare discriminative strength of individual signals in compare to baseline signals.

Finally, due to the large number of experiments there are displayed evaluation metric results with different LTR algorithms only if we found out that the new tested signals are strong sufficiently. Moreover if the computed signal importances are low for given set of new signals, we can assume that the values of evaluation metrics in extended datasets by these signals do not increase at all.

4.5 Handcrafted signals experiments

Within the experiments we divided handcrafted signals to three different groups.

Table 4.3: Handcrafted I signals

Signal ID	Signal name
1000	avgAnchorTextLength
1001	avgWordCount
1002	avgDivsBefore
1003	avgWordLength
1004	avgTagsBefore
1005	bodySize
1006	avgTagsInside

4.5.1 Handcrafted I signals

List of the signals from first group is presented in Table 4.3. The handcrafted I signal importance are depicted in Figure 4.9, where it can be seen that

the strongest signal represents document body size. Another good signals according to the results from random forest seem to be average number of tags before each paragraph in document and average number of tags inside each paragraph in document.

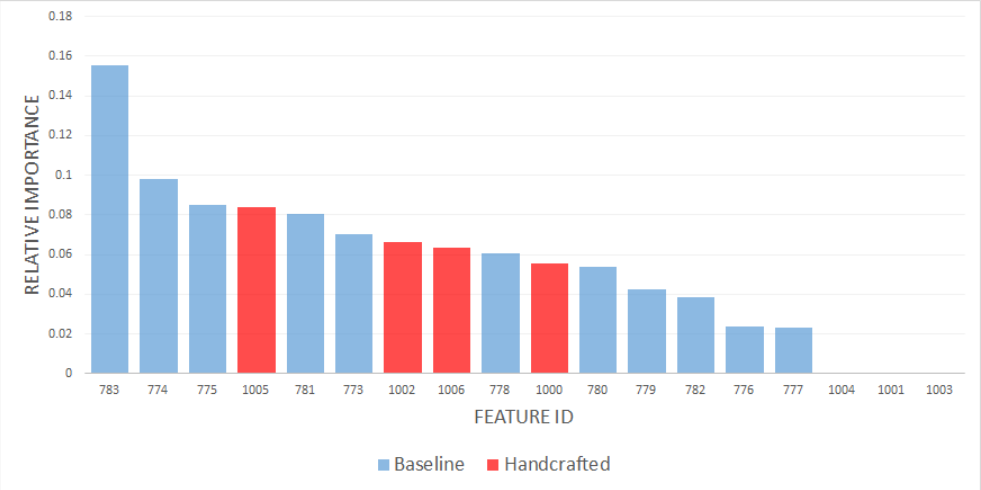


Figure 4.9: Handcrafted I - Relative signal importances

Table 4.4: Handcrafted II signals

Signal ID	Signal name
1000	tokens are in top 1
1001	tokens are in top 5
1002	tokens are in top 10
1003	tokens are in doc
1004	number of tokens in doc
1005	inverted best match

4.5.2 Handcrafted II signals

The second group is called Handcrafted II signals 4.4 that represents signals dependent both on documents and on queries. Figure 4.10 shows relative importance for the tested signals in compare to text baseline ones. The strongest signal is inverted best match 2.2 and the second is number of tokens in doc. We can see the rest of tested signals has just negligible value of importances since it strongly correlates each other.

We conducted several other tests for this group of signals and computed the same signals for different part of documents, such as URL, headers, titles, content and all together. The results given in Table 4.5 show, especially for

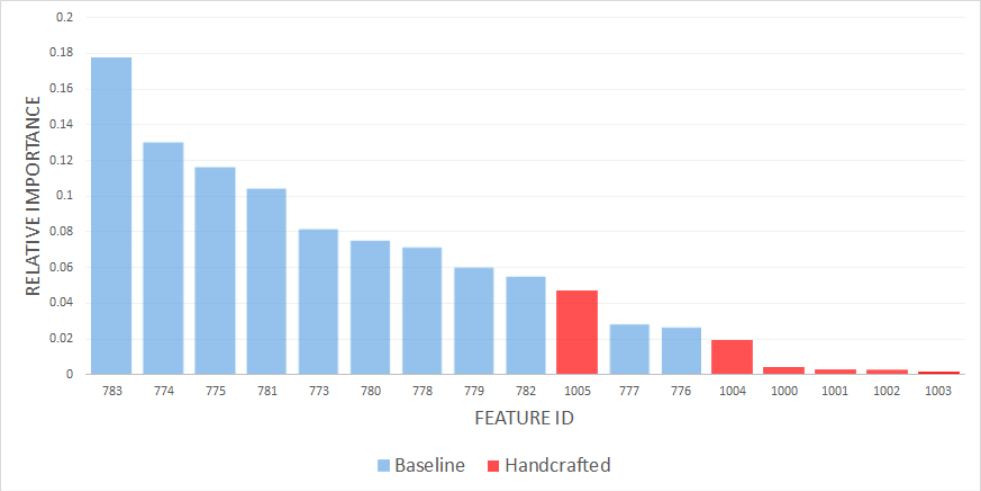


Figure 4.10: Handcrafted II - Relative signal importances

Number of Tokens in Doc and Inverted Best Match signal, that it is the best choice to combine all parts from document together. In other words, the bigger text content, the better relative signal importance we got.

Table 4.5: Handcrafted II - Comparison of signal quality crafted from different parts of documents

Document part	**1000**	**1001**	**1002**	**1003**	**1004**	**1005**
All	0.42%	0.28%	0.24%	0.13%	1.91%	4.71%
Content	0.31%	0.26%	0.24%	0.24%	1.67%	3.99%
Title	0.45%	0.23%	0.14%	0.14%	1.61%	2.60%
Headers	0.36%	0.26%	0.22%	0.23%	1.49%	2.39%
URL	0.38%	0.26%	0.20%	0.21%	1.19%	2.12%

4.5.3 Handcrafted III signals

The last batch of tests was conducted over the Handcrafted III signals 4.6 that cover only one type of signal - Query token density in document 2.1. We provide several experiments with different types of documents as well as for the previous group of signals. The experimental results in Table 4.7 confirm that the combination of all available text data gives the best relative importance results.

Furthermore, we can see that the Query token density signal has the highest relative importance in comparison with text baseline signals. The graph 4.11 shows relative importances compared with all baseline signals. We

Table 4.6: Handcrafted III signals

Signal ID	Signal name
1000	Query token density

can observe that our tested signal is eleventh strongest signal according to random forest method.

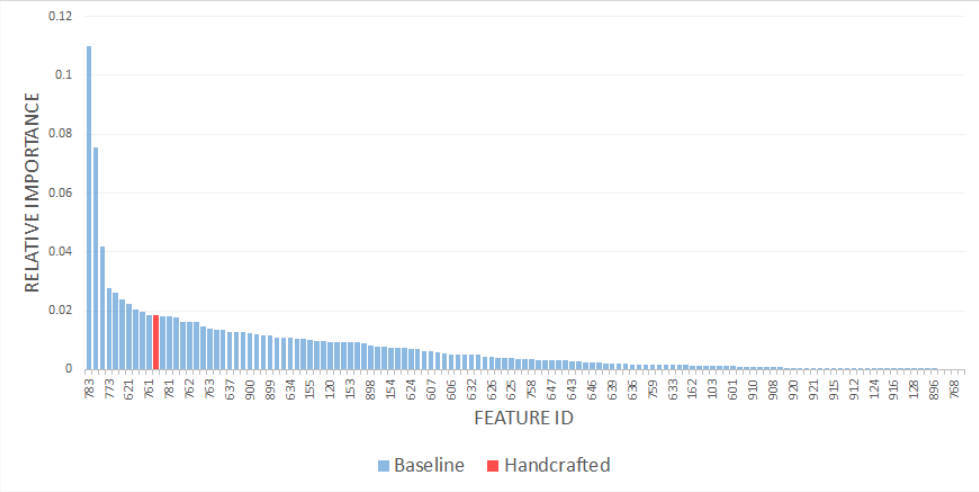


Figure 4.11: Handcrafted III - Relative signal importances compared with all baseline signals

Table 4.7: Handcrafted III - Comparison of signal quality crafted from different parts of documents

Document part	1000
All	10.22%
Content	7.62%
Header	5.57%
Title	5.52%
URL	4.50%

4.5.4 Complete Handcrafted signals

The final group of tests focus on all handcrafted signals and compares their importance to each other. The results are given in Figure 4.12 on which we can see dominance of the Document body size and Document Token Density signals.

Table 4.9 shows evaluation performance for several LTR algorithms learned by different sets of signals. Text baseline dataset gives significantly better results for all LTR algorithm and each evaluation metrics, nevertheless we can see that our handcrafted signals have considerably better results than results given by RandomRank that just sorts document in utterly random order so there is an evidence that handcrafted signals can improve document ranking as well.

For each tested LTR algorithm was monitored usage of signals during the learning phase, so it is possible to determine the importances of individual signals even for specific LTR algorithm. We can observe signal importances computed for AdaRank in Table, for LambdaMART in Table and finally for RcRank in Table. Seznam's RcRank has own signal importance metric called `nutricity` that represents how often a signal is used inside the algorithm and how well divides documents. Seznam uses also `potential` which is similar to `nutricity`, nevertheless unlike `nutricity` it describe potential value of `nutricity` if the signal would had been used.

Table 4.8: Complete handcrafted signals

Signal ID	Signal name
1000	tokens are in top 1
1001	tokens are in top 5
1002	tokens are in top 10
1003	tokens are in doc
1004	number of tokens in doc
1005	inverted best match
1006	doc token density
1007	avgAnchorTextLength
1008	avgWordCount
1009	avgDivsBefore
1010	avgWordLength
1011	avgTagsBefore
1012	bodySize
1013	avgTagsInside

4.6 Vector space model signals experiments

In this section we provide exhaustive results from experiments with Vector Space Model signals, such as TF-IDF, LSI and LDA. Individual experiments were grouped by specific models, nevertheless each tested dataset respects the same structure of extended signals. For each experimental test we modeled signals from different parts of documents. Specific structure is depicted in Table 4.10.

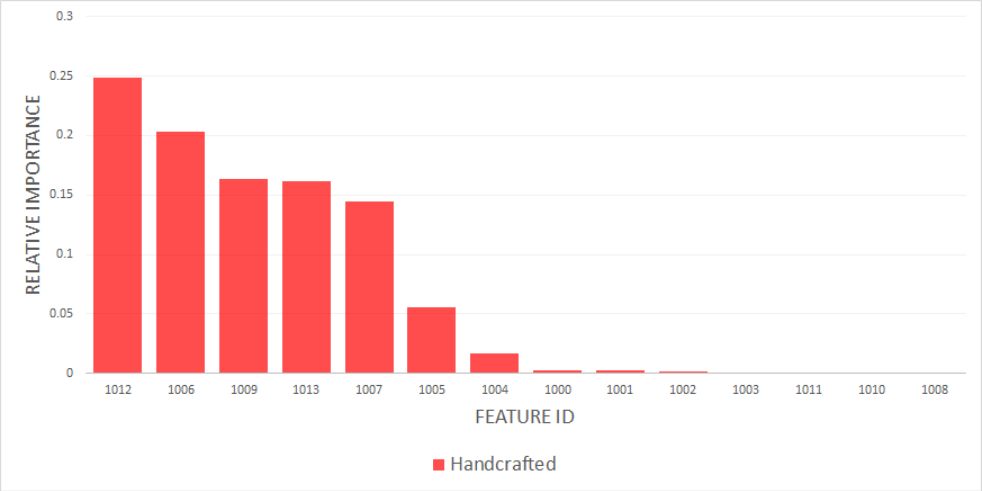


Figure 4.12: Comparison of Handcrafted relative importance signals

Table 4.9: Handcrafted signals evaluation results

LTR Algorithm	Signal dataset	NDCG@10	ERR@10	SR@20
AdaRank	Text baseline	0.58062648	0.33272698	0.44164857
AdaRank	Handcrafted	0.4688737	0.27236293	0.424339
AdaRank	Text baseline + Handcrafted	0.580423	0.33423424	0.44353423
LambdaMART	Text baseline	0.59855354	0.34760549	0.45648788
LambdaMART	Handcrafted	0.47786243	0.27104893	0.430321
LambdaMART	Text baseline + Handcrafted	0.59634342	0.34235239	0.45634526
RcRank	Text baseline	0.59139023	0.3096874	0.43420696
RcRank	Handcrafted	0.4746935	0.27685408	0.500645
RcRank	Text baseline + Handcrafted	0.5945345	0.304534636	0.43986342
RandomRank	-	0.386	0.20236293	0.392123

Table 4.10: VSM signals

Signal ID	Signal name
1000	all
1001	content
1002	header
1003	title
1004	URL

Moreover, for each VSM we found the combination of parameters bringing the best results. There were tested parameters such as size of dictionary, preprocessing type and (only for LSI and LDA models) number of topics.

4.6.1 TF-IDF

First experiment with the VSM TF-IDF signals is presented in Table 4.11 and shows signal relative importance dependency on chosen pre-processing type. We computed each preprocessing on corpuses with the same dictionary length equal to one million words. We can observe that none pre-processing gives the best results, further observation is that lemmatization gives better results than both variants of stemming.

Table 4.11: VSM Signal TF-IDF - Preprocessing dependency on relative signal importance

Preprocessing type	**1000**	**1001**	**1002**	**1003**	**1004**
Aggressive stemming	2.47%	1.91%	1.73%	2.29%	1.94%
Stemming	3.34%	2.51%	2.25%	3.09%	2.66%
Lemmatization	5.46%	3.10%	2.49%	3.31%	3.97%
None	6.49%	3.19%	2.19%	2.84%	4.47%

Due to the results from previous experiment we did not chose any pre-processing method and further, we focused on the importance of the signal depending on the size of dictionaries from which was created test corpora. Results from this experiment are proposed in Table 4.12 and show that the larger dictionaries then 500k words have negligible effect on the value of signal relative importance, notwithstanding the size of dictionary is really important parameter as signals crafted from small dictionaries have practically no relative importance.

Table 4.12: VSM Signal TF-IDF - Dictionary size dependency on relative signal importance

Dictionary size	**1000**	**1001**	**1002**	**1003**	**1004**
100k	0.07%	0.08%	0.00%	0.00%	0.00%
500k	5.38%	3.04%	2.48%	3.31%	4.00%
1M	5.47%	3.11%	2.50%	3.31%	3.96%
2M	5.49%	3.13%	2.49%	3.31%	3.95%

Figure 4.13 shows relative importances of our TF-IDF signals compared with text baseline ones. We set the best configuration of parameters according to previous experiments.

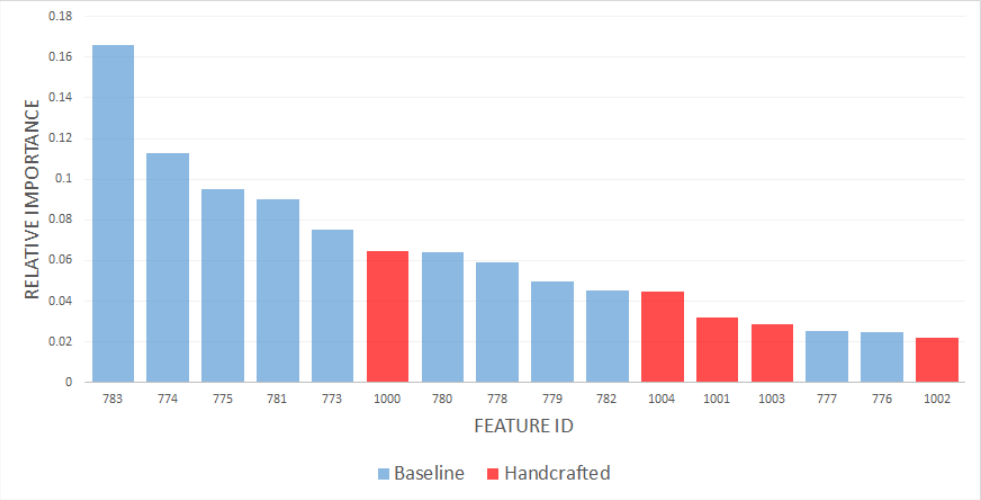


Figure 4.13: VSM TF-IDF - Relative signal importances compared to text baseline signals

4.6.2 LSI

Table 4.13: VSM Signal LSI - Preprocessing dependency on relative signal importance

Preprocessing type	**1000**	**1001**	**1002**	**1003**	**1004**
Aggressive stemming	4.39%	3.91%	3.49%	4.00%	4.09%
Stemming	4.20%	3.79%	3.61%	4.33%	4.43%
Lemmatization	6.56%	4.89%	4.74%	5.83%	5.83%
None	7.92%	5.10%	4.99%	6.03%	6.41%

As in the case of TF-IDF signals we conducted several experiments for LSI signals. First one, showed in Table 4.13, represents dependency on pre-processing type. There is an evidence that the best pre-processing to use in case of LSI is none pre-processing or lemmatization. Second one, showed in Table 4.14, describes dependency on dictionary length. We can observe the highest value of relative importances for dictionary size equal to 2 millions, nevertheless the quality gap between 500 thousands and 2 millions dictionary is just negligible.

The Table 4.15 refers to experiment that focuses on the dependency between signal importance and number of topics in LSI spaces. In our experiment we can see that the more topics gives slightly higher importance, however importance gap is very small and number of topics cannot be considered as significant parameter. There is an assumption that in the document text is much less than 50 discriminative topics.

Table 4.14: VSM Signal LSI - Dictionary size dependency on relative signal importance

Dictionary size	1000	1001	1002	1003	1004
100k	0.07%	0.08%	0.00%	0.00%	0.00%
500k	5.38%	3.04%	2.48%	3.31%	4.00%
1M	5.47%	3.11%	2.50%	3.31%	3.96%
2M	5.49%	3.13%	2.49%	3.31%	3.95%

Table 4.15: VSM Signal LSI - Number of topic dependency on relative signal importance

Dictionary size	1000	1001	1002	1003	1004
50	7.65%	5.12%	4.96%	6.06%	6.41%
100	7.92%	5.10%	4.99%	6.03%	6.41%
200	8.14%	5.17%	4.94%	6.08%	6.36%

Figure 4.15 shows relative importances of our LSI signals compared with text baseline ones. We set the best configuration of parameters according to previous experiments.

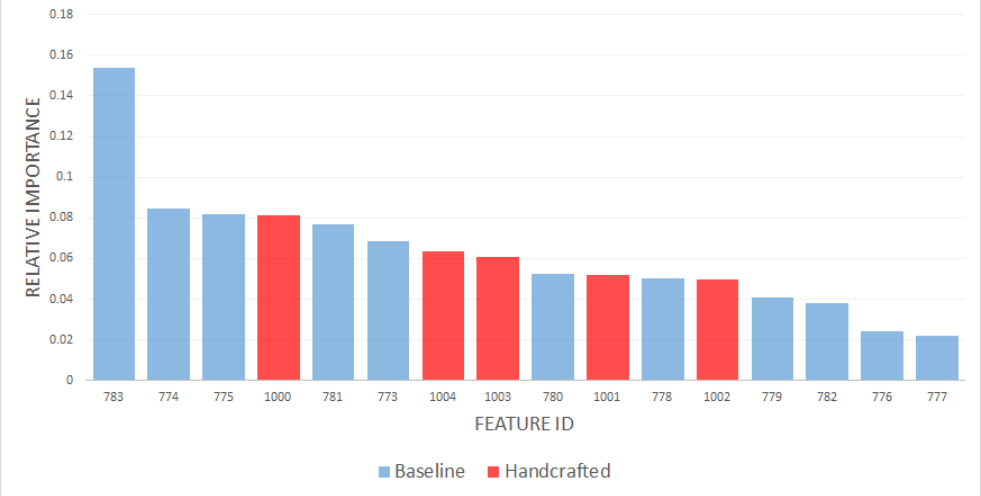


Figure 4.14: VSM LSI - Relative signal importances compared with text baseline signals

4.6.3 LDA

First experiment given in Table 4.16 depicts dependency between pre-processing type and relative importances of tested signals. The best results are provided

by none pre-processing as in previous experiments with TF-IDF or LSI spaces.

Table 4.16: VSM Signal LDA - Preprocessing dependency on relative signal importance

Preprocessing type	**1000**	**1001**	**1002**	**1003**	**1004**
Aggressive stemming	2.48%	2.12%	1.46%	2.11%	1.68%
Stemming	3.19%	2.54%	1.97%	2.89%	2.30%
Lemmatization	5.20%	3.67%	3.15%	4.49%	3.35%
None	6.30%	4.06%	2.71%	3.21%	0.70%

Table 4.17 shows signal importances for different size of dictionaries. There was reached the same results as in case of TF-IDF or LSI signals, therefore it is very small quality gap between dictionary size equals to a half million and 2 millions. As the best results we can then considered dictionary size equal 500k.

Table 4.17: VSM Signal LDA - Dictionary size dependency on relative signal importance

Dictionary size	**1000**	**1001**	**1002**	**1003**	**1004**
100k	4.05%	4.05%	3.73%	2.95%	3.12%
500k	6.26%	5.48%	4.74%	5.88%	4.63%
1M	6.26%	5.48%	4.78%	5.87%	4.73%
2M	6.25%	5.45%	4.77%	5.91%	4.74%

Table 4.18: VSM Signal LDA - Number of topic dependency on relative signal importance

Dictionary size	**1000**	**1001**	**1002**	**1003**	**1004**
50	6.80%	5.89%	4.68%	4.53%	2.86%
100	6.30%	4.06%	2.71%	3.21%	0.70%
200	6.60%	3.83%	2.60%	3.11%	0.45%

Figure 4.15 shows relative importances of our LDA signals compared with text baseline ones. We set the best configuration of parameters according to the previous experiments.

Table 4.19 describes results for given evaluation metrics and LTR algorithms. We can see that our VSM signals with the best configurations do not outperform text baseline signals, nevertheless our VSM signals help to LTR algorithm to rank the documents much better than it would have been just ranked in random way.

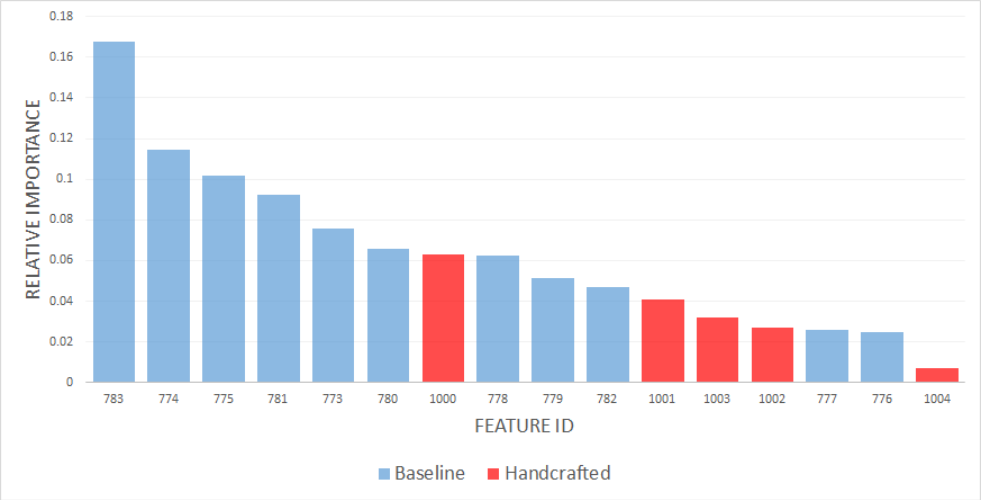


Figure 4.15: VSM LDA - Relative signal importances compared with text baseline signals

Table 4.19: Vector Space Model signals evaluation results

LTR algorithm	**Signal dataset**	**NDCG@10**	**ERR@10**	**SR@20**
AdaRank	Text baseline	0.58062648	0.332727	0.441649
AdaRank	VSM signals	0.4563223	0.272342	0.423435
AdaRank	Text baseline + VSM signals	0.582312	0.325465	0.442345
LambdaMART	Text baseline	0.59855354	0.347605	0.456488
LambdaMART	VSM signals	0.462324	0.273435	0.435234
LambdaMART	Text baseline + VSM signals	0.59212324	0.342326	0.452322
RcRank	Text baseline	0.59139023	0.309687	0.434207
RcRank	VSM signals	0.47123154	0.269834	0.472342
RcRank	Text baseline + VSM signals	0.59234234	0.310234	0.434234
RandomRank	-	0.386	0.202363	0.392123

4.7 Semantic signals experiments

In this section we provide several experiments over the last group of extended text signals. As mentioned above, within this text signal group we have created Simhash text signals and Semantic hashing text signals.

4.7.1 Semantic hashing

Table 4.20 assigns feature ID to semantic hashing signal. For the initial experiments, in which we are finding the ideal settings of neural networks, use only one signal at a time.

Table 4.20: Semantic hashing signals

Signal ID	Signal name
1000	Semantic hashing

In Table 4.21 we can observe several tests with different parameters of SAE neural networks. We experimented with different input unit size, different level of noise at input and hidden layers, number of layers (counted from the input layer to the middle (latent) layer) and size of middle units (result hashed vector). After a series of measurements we can observe that letter 3gram with vector input size equals to 2000, input and hidden noise equals to 15%, number of layers equals to 3 and latent layer size equals to 128 gives the best relative importances to text baseline signals.

Table 4.21: Semantic hashing - Relative importance with different parameters

Type	Vector size	Noise	# layers	# middle units	Relative importance
Letter 3gram	2000	15%	3	128	11.55%
Letter 3gram	2000	0%	3	128	11.50%
TOP freq BoW	5000	15%	3	128	11.46%
Letter 3gram	2000	15%	3	64	11.41%
TOP freq BoW	5000	15%	5	128	11.41%
TOP freq BoW	5000	15%	4	128	11.41%
TOP freq BoW	5000	0%	3	128	11.39%
Letter 2gram	2000	15%	3	128	11.38%
Letter 3gram	2000	35%	3	128	11.24%
Letter 3gram	5000	15%	3	128	11.21%
TOP freq BoW	5000	35%	3	128	11.17%
Letter 3gram	10000	15%	3	128	11.10%
Avg W2V	2000	15%	3	128	8.84%
Top freq TF-IDF	2000	15%	3	128	8.67%

Further it can be observed need to add at least a small amount of noise during the neural network training phase. However, too much noises bring worse results then none noise. Regarding the size of the input vector it can be seen the best choice of size about 2000. Higher dimension of input vectors brings worse results.

We also have tried to reduce the size of the latent (middle) space to 64, nevertheless the value of relative importance went down.

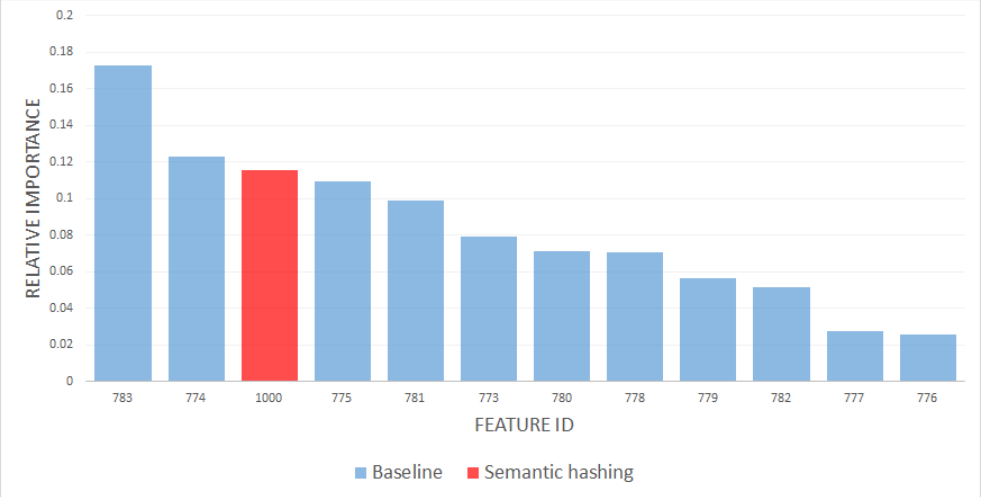


Figure 4.16: Semantic hashing - Relative signal importances compared to text baseline signals

On the Figure 4.16 are displayed Semantic Hashing signal quality measured by relative importances (the best configuration according to previous experiments) compare to text baseline signals while Figure 4.17 shows results with semantic hashing signal in compare to all baseline signals.

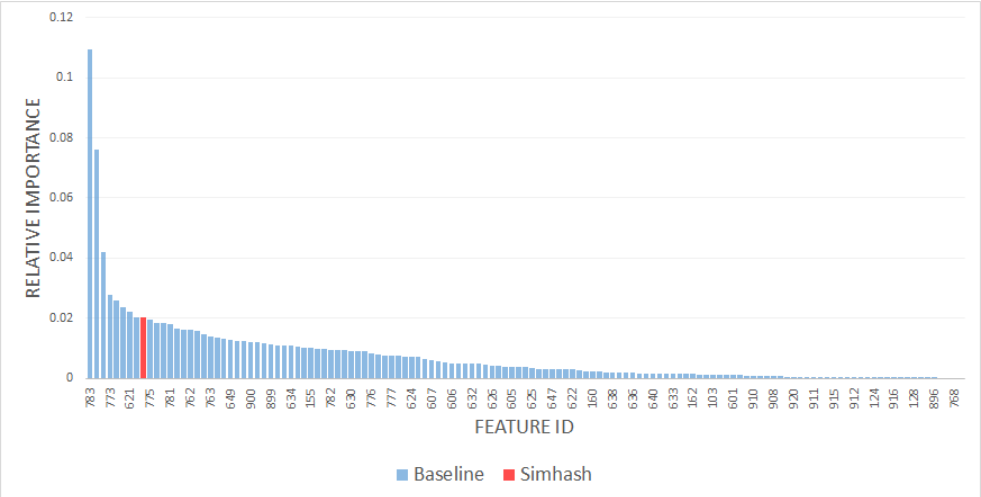


Figure 4.17: Semantic hashing - Relative signal importances compared to all baseline signals

Following graphs show relative importances according to individual LTR algorithms. We can see that AdaRank results (see Figure 4.18) are not so good in compare to LambdaMART (see Figure 4.19), where our signal is fifth strongest signal.

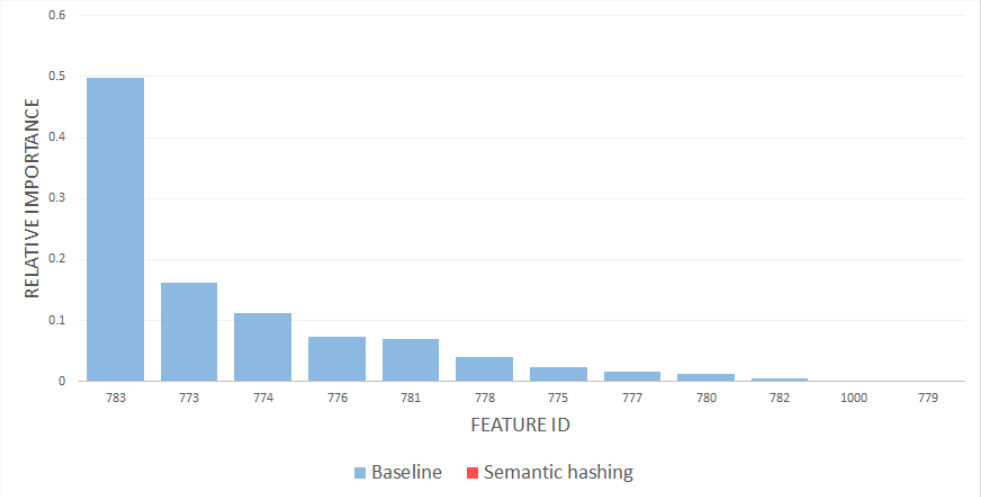


Figure 4.18: Semantic hashing - Relative signal importances compared to text baseline signals according to AdaRank LTR algorithm

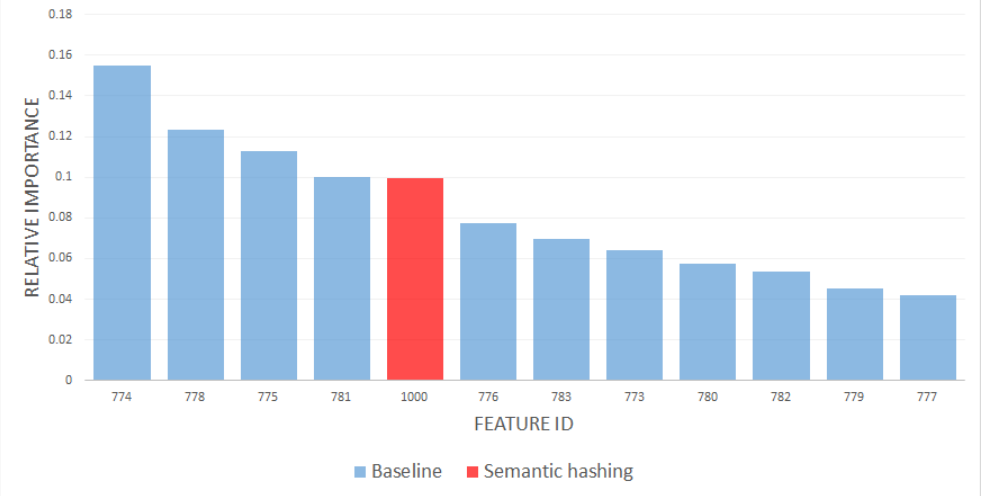


Figure 4.19: Semantic hashing - Relative signal importances compared to text baseline signals according to LambaMART LTR algorithm

Figure 4.20 depicts relative importances according to Seznam's RcRank LTR algorithm. They use special own measures to determine which signal

is stronger than another one. Nutricity is the real measure, which indicates how often and how well the signals were selected during training phase of RcRank algorithm. Potential is just a potential measure, which represents a hypothetical signal quality if the signal would be used inside the RcRank.

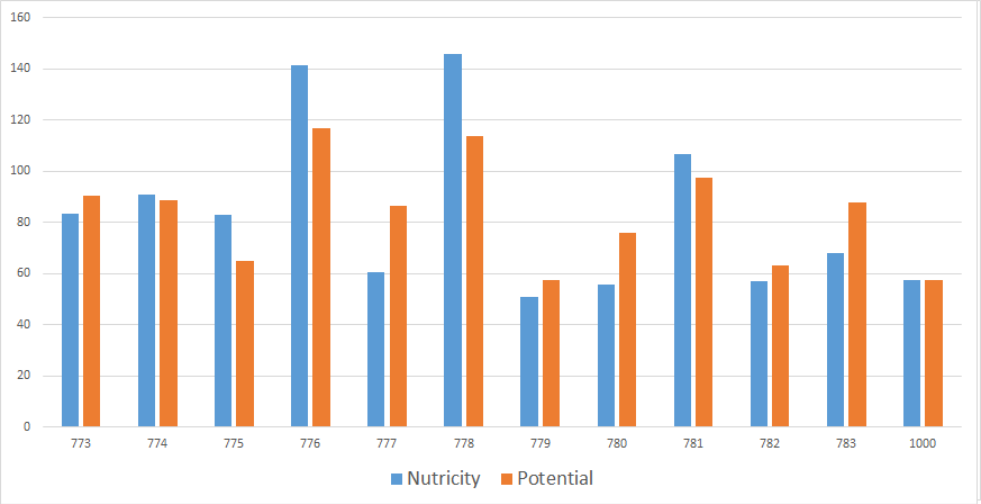


Figure 4.20: Semantic hashing - Relative signal importances compared to text baseline signals according to RcRank LTR algorithm

4.7.2 Simhash

This section is focused on the Simhash signal testing and evaluation. Table 4.22 shows Simhash signal mapping to the specific ID for better comprehensibility in following charts and result tables.

Table 4.22: Simhash signals

Signal ID	Signal name
1000	Simhash

Simhash signal is third strongest signal in compare to text baseline signal with more than 11 % relative importances (see Figure 4.21) and ninth strongest signal in compare to all 130 baseline signals (see Figure 4.22).

Table 4.23 provides evaluation results for both Simhash and Semantic hashing signals. We computed several experiments over the AdaRank, LambdaMART, RcRank and RandomRank (which is basically sorting in random manner) and evaluated via several evaluation LTR measures. We can observe as well as previous groups of signals that semantic signals do not outperform

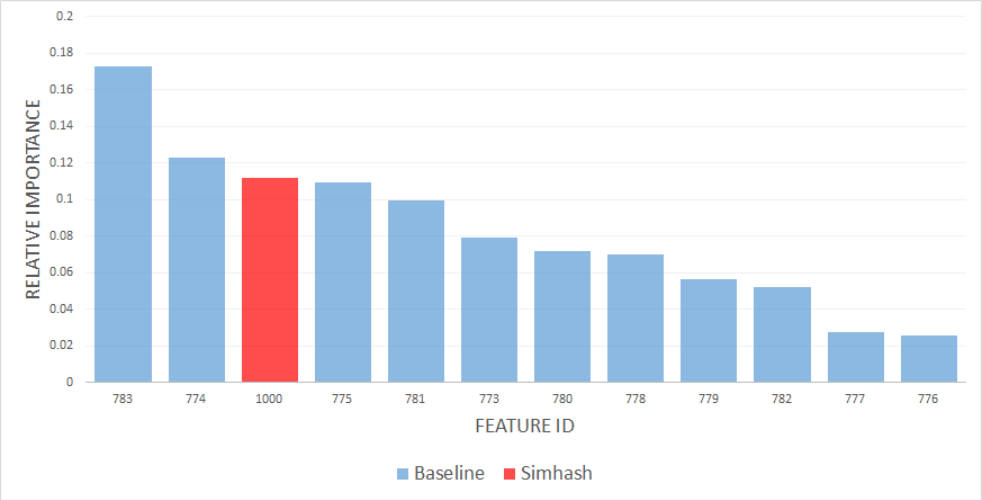


Figure 4.21: Simhash - Relative signal importances compared with text baseline signals

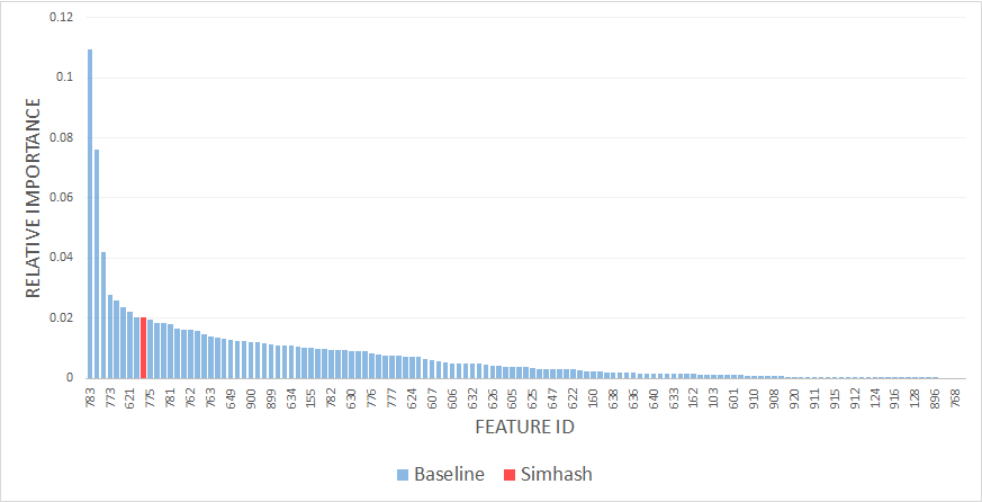


Figure 4.22: Simhash - Relative signal importances compared with all baseline signals

the results of the baseline. Nevertheless there is reached significantly better results rather than trivial random ranking documents.

Table 4.23: Semantic signals evaluation results

LTR algorithm	Signal dataset	NDCG@10	ERR@10	SR@20
AdaRank	Text baseline	0.58062648	0.332727	0.441649
AdaRank	Semantic signals	0.4761423	0.298534	0.435663
AdaRank	Text baseline + Semantic signals	0.581234	0.32834	0.44023
LambdaMART	Text baseline	0.59855354	0.347605	0.456488
LambdaMART	Semantic signals	0.472345	0.288242	0.436264
LambdaMART	Text baseline + Semantic signals	0.599432	0.347235	0.450455
RcRank	Text baseline	0.59139023	0.309687	0.434207
RcRank	Semantic signals	0.473245	0.278342	0.46894
RcRank	Text baseline + Semantic signals	0.5902359	0.309845	0.43356
RandomRank	-	0.386	0.202363	0.392123

4.8 Comparison of all generated signals

This section describes results with all generated signals in one dataset together. There were chosen just couples of the best signals (with the best configuration) from each aforementioned signal groups and its description is given in Table 4.24.

Table 4.24: Complete list of generated signals

Type	Signal ID range
Handcrafted	1004 - 1010; 1012 - 1018
VSM	1001 - 1003
Semantic hashing	1000
Simhash	1011

We can see that the the newly added signals are re-numbered from 1000, we added just one signal created by Simhash and Semantic hashing, three signals from VSM, such as LSI, LDA and TF-IDF and finally several handcrafted signals.

Figure 4.23 shows mutual relative importances of all extended signals. According to the graph we can consider first four signals as the most important. It is a Simhash signal, Semantic hashing and two Handcrafted signals, more specifically Document Token Density and Document Body size signal.

Unlike the previous figure, following ones depict relative signal importance in compare to the different baseline signals. (See 4.24, 4.25, 4.26)

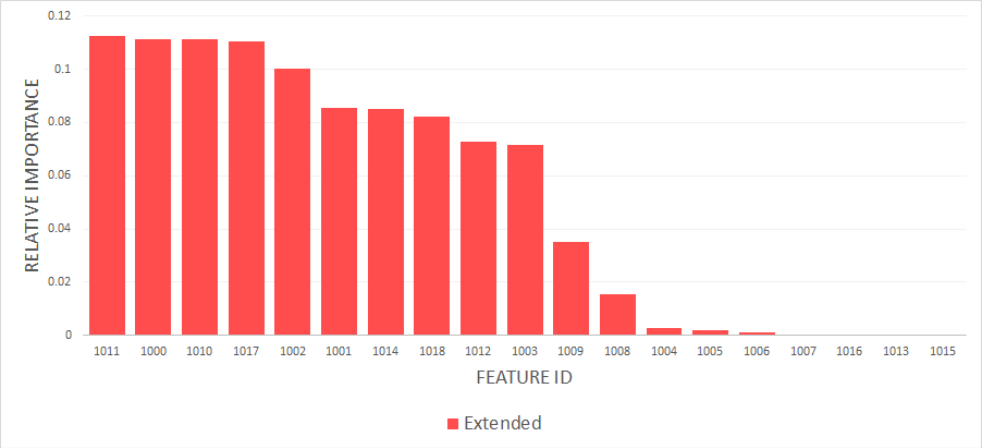


Figure 4.23: Text signals - Relative mutual importances

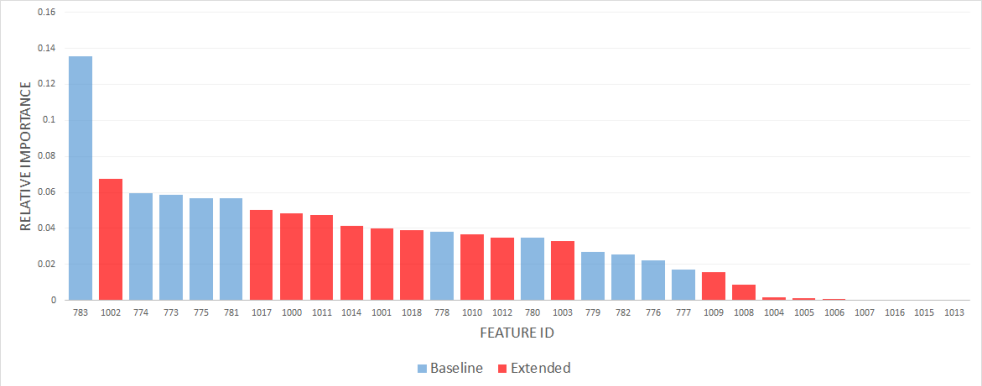


Figure 4.24: Text signals - Relative mutual importances in compare to text baseline

In the Figure 4.24 we can observe different order of importance for extended signals than was given in Figure 4.23. In compare to text baseline signals seems to be the most important VSM LSI signal. It is because of baseline text signals correlates with handcrafted and semantic signals.

The Importance of extended signals is relatively high in comparison to all baseline signals (see 4.26) and without text baseline signals (see 4.25). We can see the LSI signal in TOP 5 the most important signal ever, nevertheless Simhash and Semantic hashing signals have relatively high importance as well.

Table 4.25 describes evaluation results over several LTR algorithms and evaluation metrics. We can see, as in previous measurements, that our signals do not outperform provided baseline by Seznam.cz. Nevertheless a little improvement by combining all the extended signals was reached, especially

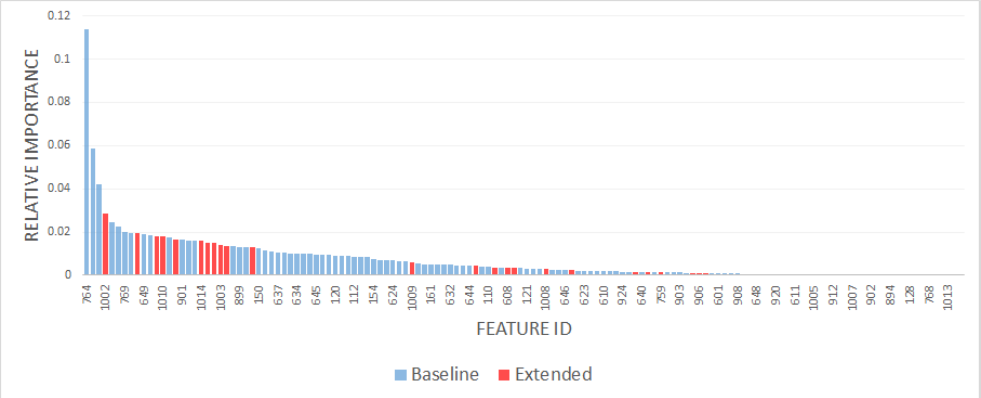


Figure 4.25: Text signals - Relative mutual importances in compare to without text baseline

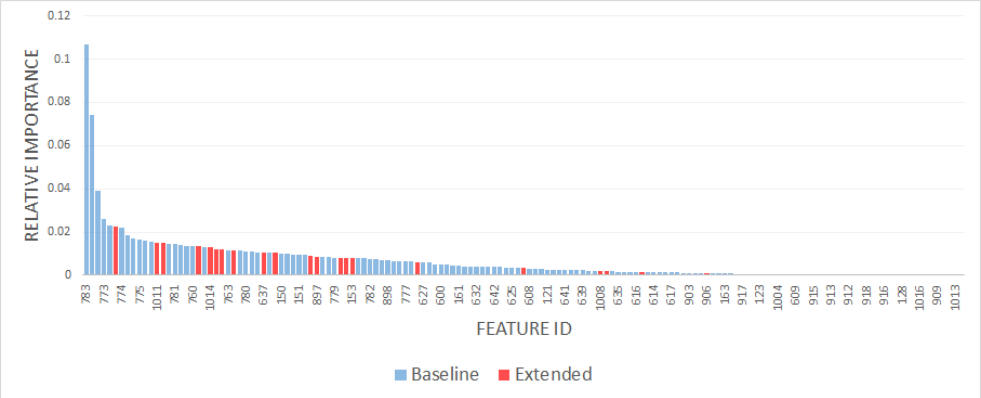


Figure 4.26: Text signals - Relative mutual importances in compare to all baseline

NDCG@10 and ERR@10 measures have increased in comparison to just VSM models in Table 4.19, Handcrafted in Table 4.9 and Semantic signals in Table 4.23.

Unfortunately adding our extended text signals to the baseline signals do not increase any of tested evaluation measure. That means we could not find any uncorrelated text signals to provided Seznam's baseline signal dataset.

4.9 Query expansion experiments

The experimental results with query expansion are provided in following section. As above mentioned, we have tried to expand individual queries by Word2Vec model. We assumed that the closest vectors to given word com-

Table 4.25: All extended signals evaluation results

LTR algorithm	Signal dataset	NDCG@10	ERR@10	SR@20
AdaRank	Text baseline	0.58062648	0.332727	0.441649
AdaRank	Extended signals	0.49362333	0.297134	0.43019
AdaRank	Text baseline + Extended signals	0.58065162	0.333833	0.442523
LambdaMART	Text baseline	0.59855354	0.347605	0.456488
LambdaMART	Extended signals	0.50553207	0.286831	0.433017
LambdaMART	Text baseline + Extended signals	0.59745315	0.345288	0.460368
RcRank	Text baseline	0.59139023	0.309687	0.434207
RcRank	Extended signals	0.507241	0.2872352	0.4409134
RcRank	Text baseline + Extended signals	0.5924235	0.31053154	0.4329641
RandomRank	-	0.386	0.202363	0.392123

puted by any distance measure represent semantically the most similar word (e.g. synonyms).

Nevertheless our training dataset contained only one and half millions of documents (our dataset from Seznam.cz). According to [38] there is needed to use much larger corpora (billions of words) to train word2vec model for better results.

Therefore, our trained word2vec model does not return satisfactory words. We have tried to expand each query token on the top 5 closest words given the word2vec space and for example for given query token "myslivost" we got semantically totally different words.

For these reasons our query expansion did not bring any evaluation or importance improvements to particular signals. Nevertheless following Chapter 5 provides possible suggestions to improve query expansion.

Chapter 5

Future work

In this section we will focus on the possibilities of further approaches to find improving text signals relevance for full text search.

First of all it could be interesting try to use doc2vec [39] approach to generate new text signals. Doc2vec (also known as paragraph2vec or sentence embeddings) modifies the word2vec algorithm to unsupervised learning of continuous representations for larger blocks of text, such as sentences, paragraphs or entire documents. Moreover, using this approach it would be possible to compare document and query directly without any averaging (case of word2vec).

Second, another suggestion to new text signal would be Supervised Semantic Hashing (SSH). SSH defines a class of nonlinear models that are discriminatively trained to directly map from the word content in a query-document to a ranking score. Unlike LSI the SSH model is trained from a supervised signal directly on the ranking task of interest, which according to authors results [40] seems to be meaningful advantage. In other words, SSH represents models in which every document-document or document-query token pair represents independent signal. Each signal has assigned own weight which is adjusted during the training phase. Result signal from SSH would represent ranking value given for particular query and document from the SSH model.

Third, as already mentioned in section 4.9 there is needed to use larger dataset to learn word2vec model to return semantically similar words for given query tokens. Although many public datasets is available, unfortunately we need for this task czech datasets which are not publicly available.

Fourth, within this work we did not use any sophisticated query expansion such as query correction, re-weighting the terms in the original query, etc. There are also more sophisticated system to get synonyms for given query tokens (e.g. WordNet [41]).

Finally, it could be also very interesting to combine Full-text search with the document summarization task. We can consider the query as an user

attempt to create a their subjective document summarization. Therefore there is an assumption that if we create summarization for all documents then we will be able to get better results with matching queries with these document summarizations.

Conclusion

In this work we have reviewed and analyzed major LTR algorithms and furthermore, we implemented own AdaRank algorithm with different weak ranker. Within experiments with the AdaRank algorithm we have showed that the biggest limitations are found in used signals, not in the used LTR algorithms.

Within the thesis we also reviewed and analyzed metrics for relevance ranking and chose main three representatives of evaluation metrics to our designed and implemented system for testing and evaluation of new signals.

Analysis and review were also performed over the commonly used text signals known from literature. We focused mainly on text signals used by Google search engine and semantic signals in general, such as LSI, LDA spaces, Semantic hashing approach using Deep learning networks or Simhash which is primarily used to detect the same documents in large text streams.

Although our designed and implemented system for testing and evaluation of new signals consists several bash scripts prepared to be executed on compute nodes in Metacentrum due to the large computing resources, its usage is really simple and straightforward. To test and evaluate a new signals there is needed just baseline dataset in the appropriate format and text signal file in the appropriate format, everything else is fully automated. All scripts and implementation are available for all staff and students from the university in Gitlab server [6]. For further information about the system follow the user manual.

There were designed and implemented new text signals according to analysis which were tested and evaluated via our system afterwards. We have conducted bunch of experiments that brought several interesting conclusions. Since we divided our experiments to the three groups, we divide our conclusion to the same three groups as well.

[6] available from https://gitlab.fit.cvut.cz/hnizdja2/diplomathesis

Conclusions from Handcrafted signal experiments

- signals with non-binary values have typically higher importance
- the most important signals are Document body size, Inverted best match 2.2 and Query token density 2.1

Conclusions from Vector Space Model signal experiments

- Signals without any special pre-processing method have the best evaluation results, nevertheless lemmatization is meaningfully better than stemming
- sufficient dictionary size is 500,000
- the best results was reached by LSI with 200 topics
- LSI and LDA outperform TF-IDF

Conclusions from Semantic hashing signal experiments

- TF-IDF input vectors is not suitable for Deep learning training since the values are too small and neural networks do not train properly; simple using of BoW vectors brings significantly better results
- letter ngram approach outperform classical approach to choose the most frequent word in task of reducing input vector space

Final conclusion

Our extended text signals did not outperform provided baseline text signals by Seznam, nevertheless combining all of these signals we got reasonable results by all of tested evaluation measures. Unfortunately adding our extended text signals to the baseline signals do not improve any evaluation results since we could not find any uncorrelated signals with the baseline.

We also show (see Figure 4.6) that choice of the learning to rank algorithm is not the key aspect for ranking documents. Much more important seems to be the feature selection from the documents and datasets itself.

Bibliography

[1] Manning, C. D.; Raghavan, P.; Schütze, H. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008, ISBN 0521865719, 9780521865715.

[2] Liu, T.-Y. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, volume 3, no. 3, 2009: pp. 225–331.

[3] Breiman, L. Random Forests. *Mach. Learn.*, volume 45, no. 1, Oct. 2001: pp. 5–32, ISSN 0885-6125, doi:10.1023/A:1010933404324. Available from: `http://dx.doi.org/10.1023/A:1010933404324`

[4] Modrý, M. *Learning to Rank Algorithms*. Master's thesis, Czech Technical University in Prague, may 2014. Available from: `http://cyber.felk.cvut.cz/research/theses/papers/471.pdf`

[5] Friedman, J. H. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, volume 29, no. 5, 10 2001: pp. 1189–1232, doi: 10.1214/aos/1013203451. Available from: `http://dx.doi.org/10.1214/aos/1013203451`

[6] Burges, C. J. C. From RankNet to LambdaRank to LambdaMART: An Overview. Technical report, Microsoft Research, 2010. Available from: `http://research.microsoft.com/en-us/um/people/cburges/tech_reports/MSR-TR-2010-82.pdf`

[7] Herbrich, R.; Graepel, T.; Obermayer, K. Large Margin Rank Boundaries for Ordinal Regression. In *Advances in Large Margin Classifiers*, edited by A. Smola; P. Bartlett; B. Schölkopf; D. Schuurmans, Cambridge, MA: MIT Press, 2000, pp. 115–132.

[8] Burges, C.; Shaked, T.; Renshaw, E.; et al. Learning to Rank Using Gradient Descent. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, New York, NY, USA: ACM, 2005, ISBN

1-59593-180-5, pp. 89–96, doi:10.1145/1102351.1102363. Available from: http://doi.acm.org/10.1145/1102351.1102363

[9] Mitchell, T. M. *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., first edition, 1997, ISBN 0070428077, 9780070428072.

[10] Herbrich, R.; Graepel, T.; Obermayer, K. *Large margin rank boundaries for ordinal regression*. MIT Press, Cambridge, MA, 2000. Available from: http://citeseer.ist.psu.edu/contextsummary/1891774/0

[11] Xu, J.; Li, H. AdaRank: A Boosting Algorithm for Information Retrieval. In *SIGIR*, Amsterdam, The Netherlabds, 2007.

[12] Freund, Y.; Iyer, R.; Schapire, R. E.; et al. An Efficient Boosting Algorithm for Combining Preferences. *J. Mach. Learn. Res.*, volume 4, Dec. 2003: pp. 933–969, ISSN 1532-4435. Available from: http://dl.acm.org/citation.cfm?id=945365.964285

[13] Chapelle, O.; Zhang, Y. Expected Reciprocal Rank for Graded Relevance. In *CIKM'09, NOVEMBER 26, 2009, HONG KONG, CHINA.*, ACM, 2009.

[14] Yahoo! Learning to Rank Challenge -. Available from: http://learningtorankchallenge.yahoo.com/

[15] Page, L.; Brin, S.; Motwani, R.; et al. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, November 1999, previous number = SIDL-WP-1999-0120. Available from: http://ilpubs.stanford.edu:8090/422/

[16] Dean, B. Googles 200 Ranking Factors: The Complete List. *Backlinko*, 2015.

[17] Deerwester, S.; Dumais, S. T.; Furnas, G. W.; et al. Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, volume 41, no. 6, 1990: pp. 391–407.

[18] Blei, D. M.; Ng, A. Y.; Jordan, M. I. Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, volume 3, Mar. 2003: pp. 993–1022, ISSN 1532-4435. Available from: http://dl.acm.org/citation.cfm?id=944919.944937

[19] Minka, T. P. Estimating a Dirichlet distribution. Technical report, 2000.

[20] Chen, E. Introduction to Latent Dirichlet Allocation. online, aug 2011.

[21] Salakhutdinov, R.; Hinton, G. Semantic Hashing. July 2009, doi: 10.1016/j.ijar.2008.11.006. Available from: http://dx.doi.org/10.1016/j.ijar.2008.11.006

[22] Bengio, Y. Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, volume 2, no. 1, 2009: pp. 1–127. Available from: http://dblp.uni-trier.de/db/journals/ftml/ftml2.html#Bengio09

[23] Manku, G. S.; Jain, A.; Das Sarma, A. Detecting Near-duplicates for Web Crawling. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, New York, NY, USA: ACM, 2007, ISBN 978-1-59593-654-7, pp. 141–150, doi:10.1145/1242572.1242592. Available from: http://doi.acm.org/10.1145/1242572.1242592

[24] LETOR Learning to Rank for Information Retrieval Dataset. http://research.microsoft.com/en-us/um/beijing/projects/letor/, accessed: 2015-04-20.

[25] Chang, C.-C.; Lin, C.-J. LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.*, volume 2, no. 3, May 2011: pp. 27:1–27:27, ISSN 2157-6904.

[26] Protobuf Google Protocol Buffer. https://developers.google.com/protocol-buffers/, accessed: 2015-04-11.

[27] Louppe, G. Understanding Random Forests: From Theory to Practice. Technical report, University of Lige, 2014.

[28] Huang, P.-S.; He, X.; Gao, J.; et al. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*, edited by Q. He; A. Iyengar; W. Nejdl; J. Pei; R. Rastogi, ACM, 2013, ISBN 978-1-4503-2263-8, pp. 2333–2338. Available from: http://dblp.uni-trier.de/db/conf/cikm/cikm2013.html#HuangHGDAH13

[29] Bag of Words Meets Bags of Popcorn. https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-3-more-fun-with-word-vectors, accessed: 2015-04-20.

[30] Mikolov, T. *Statistical language models based on neural networks*. Dissertation thesis, Brno University of Technology, 2012.

[31] MetaCentrum Virtual organization. http://metavo.metacentrum.cz/, accessed: 2015-04-05.

[32] scikit-learn Machine learning in Python. http://scikit-learn.org/stable/, accessed: 2015-04-04.

[33] RankPy Learning to Rank with Python. https://bitbucket.org/tunystom/rankpy, accessed: 2015-04-08.

[34] RankLib. http://people.cs.umass.edu/~vdang/ranklib.html, accessed: 2015-04-08.

[35] Řehůřek, R.; Sojka, P. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[36] Theanets 0.5.3 documentation. http://theanets.readthedocs.org/en/latest/quickstart.html, accessed: 2015-04-05.

[37] Simhashing made simple. http://ferd.ca/simhashing-hopefully-made-simple.html, Nov. 2012, accessed: 2015-04-18.

[38] Mikolov, T. *word2vec - Tool for computing continuous distributed representations of words.* Google. Available from: https://code.google.com/p/word2vec/

[39] Rehurek, R. *Doc2vec tutorial.* dec 2014. Available from: http://radimrehurek.com/2014/12/doc2vec-tutorial/

[40] Bai, B.; Weston, J.; Grangier, D.; et al. Learning to rank with (a lot of) word features. *Inf. Retr.*, volume 13, no. 3, 2010: pp. 291–314. Available from: http://dblp.uni-trier.de/db/journals/ir/ir13.html#BaiWGCSQCW10

[41] Miller, G. A. WordNet: A Lexical Database for English. *Commun. ACM*, volume 38, no. 11, Nov. 1995: pp. 39–41, ISSN 0001-0782, doi: 10.1145/219717.219748. Available from: http://doi.acm.org/10.1145/219717.219748

Appendix A

Acronyms

LDA Latent Dirichlet Allocation

LSI Latent Semantic Indexing

LTR Learning to Rank

MAP Mean Average Precision

MSE Mean Squared Error

NDCG Normalized Discounted Cumulative Gain

SVD Singular Value Decomposition

SSH Supervised Semantic Hashing

WTA Winner Takes All

Appendix B

User guide

This user guide contains brief description how to use particular scripts and it provides a list of required prerequisites to run implemented system.

B.1 Prerequisites

There is a list of required python libraries to run our provided scripts:

- Scikit-learn
- RankPy
- RankLib
- Gensim
- Theanets

B.2 How to use

It is highly recommended to use our script via bash jobs placed in "script" folder (see C) on MetaCentrum server.

Please follow this instruction:

1. To run evaluation script with new text signal on MetaCentrum first you need to clone our git repository to the your MetaCentrum folder.

   ```
   git clone https://gitlab.fit.cvut.cz/hnizdja2/diplomathesis.git
   ```

2. After that you have to follow standard rule for running MetaCentrum jobs. For further information please follow README file in enclosed CD.

There is still possible to use our evaluation system without an user account on MetaCentrum, however you have to run the python scripts separately and there is not provided any support.

B.2.1 Useful tips

To run GPU script based on Theanets library in MetaCentrum is needed to use following command in bash jobs file:

```
module add cuda-x.x
```

Where `x.x` denotes actual version of Cuda library. The command import appropriate module supporting GPU computations.

Appendix C

Contents of enclosed CD

- `README` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `implementation` ..... implementation sources
    - `create_features` ..... implementation of a new signals
    - `preprocessing` ..... python pre-processing scripts
    - `protobuf_to_vec` ..... python protobuffer scripts
    - `scripts` ..... bash jobs for MetaCentrum
    - `semantic_hashing` ..... python deep nets scripts
    - `lib` ..... external libraries
      - `rankpy` ..... RankPy: Learning to Rank with Python
      - `theanets` ..... deep learning python tools
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `DT_Hnizdil_Jan_2015.pdf` ..... the thesis text in PDF format