CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR'S THESIS

Antonín Novák

Design of Probabilistic Models for Text Input Correction

Department of Cybernetics

Thesis supervisor: **Ing. Jan Šedivý, CSc.**

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 23.05.2013 ……

……
Podpis autora práce

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Antonín N o v á k

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Návrh pravděpodobnostních modelů pro opravu textových vstupů

Pokyny pro vypracování:

Při zadávání hledaných výrazů často dochází k překlepům, pravopisným chybám atd. Navrhněte systém, který dokáže tyto chyby identifikovat a navrhnout opravený dotaz.

1. Analyzujte současné metody řešení úlohy.
2. Navrhněte vhodné řešení.
3. Implementujte navrženou metodu.
4. Vyhodnoťte kvalitu implementovaného řešení.

Seznam odborné literatury:

[1] C. Whitelaw, B. Hutchinson, G. Y Chung, G. Ellis – Using a Web for Language Independent Spellchecking and Autocorrection – Australia, 2009
[2] E. Brill, R. C. Moore – An Improved Error Model for Noisy Channel Spelling Correction – Redmond, USA, 2000
[3] J. Gao, X. Li, D. Micol, Ch. Quirk, X. Sun – A Large Scale Ranker-Based System for Search Query Spelling Correction – Redmond, USA, 2010
[4] H. Duan, B. Hsu – Online Spelling Correction for Query Correction – India, 2001

Vedoucí bakalářské práce: Ing. Jan Šedivý, CSc.

Platnost zadání: do konce zimního semestru 2013/2014

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2013

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Antonín N o v á k

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Bachelor Project: Design of Probabilistic Models for Text Input Correction

Guidelines:

Users make spelling errors while entering search queries. Design a system identifying and suggesting the most probable correct query.

1. Analyze latest methods solving the problem.
2. Suggest a suitable solution.
3. Implement suggested method.
4. Evaluate the resulting quality of the new algorithm.

Bibliography/Sources:

[1] C. Whitelaw, B. Hutchinson, G. Y Chung, G. Ellis – Using a Web for Language Independent Spellchecking and Autocorrection – Australia, 2009
[2] E. Brill, R. C. Moore – An Improved Error Model for Noisy Channel Spelling Correction – Redmond, USA, 2000
[3] J. Gao, X. Li, D. Micol, Ch. Quirk, X. Sun – A Large Scale Ranker-Based System for Search Query Spelling Correction – Redmond, USA, 2010
[4] H. Duan, B. Hsu – Online Spelling Correction for Query Correction – India, 2001

Bachelor Project Supervisor: Ing. Jan Šedivý, CSc.

Valid until: the end of the winter semester of academic year 2013/2014

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

L.S.

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 10, 2013

Acknowledgement

I would like to thank my family for encouraging support during writing this thesis and my whole studies. Without them this work wouldn't be possible. Furthermore I would also like to thank my thesis supervisor for his contagious enthusiasm and his helpful professional advices.

Abstract

This thesis introduces a new algorithm for a *Search Query Spelling Correction System*. It is based on *Learning to Rank* approach and allows to use a large number of various signals leading to an improved accuracy. Its performance will be tested against the conventional solution – the noisy channel model. The new system was developed on a Czech Internet search query set, but the feature vector structure and the algorithm can be easily adapted for any other human language when sufficient data is available. We will describe the algorithm's details, the training set and other datasets that were used. In the end we will present final results.

Abstrakt

Tato práce se zabývá novým algoritmem pro úlohu *Search Query Spelling Correction*. Je založen na postupu *Learning to Rank*, který dovoluje zpracovat velké množství různých signálů zvyšujících přesnost. Výkon systému bude otestován oproti běžným metodám založených na šumovém komunikačním kanálu. Nový systém je vytvořen pro prostředí českých vyhledávačů, ale struktura signálů a vlastní algoritmus ho dovolují adaptovat pro libovolný lidský jazyk s dostatkem dat. Popíšeme detaily algoritmu, trénovací množinu a další potřebná data. Na závěr porovnáme výsledky obou systému.

Contents

List of Figures

List of Tables

List of Algorithms

1 Problem formulation

The Internet contains vast amount of information. It is critical to find the required quickly. This task is done arguably well by *search engines*. The problem arises, when the user doesn't know exact spelling of the searched term or rushes and makes a typos. This may cause the search engine will find no results.

This problem can be resolved in multiple ways. The first is commonly referred as *query correction*. This is the kind of things that you may see when searching by Google and after you hit enter key, you see a *Did you mean... ?* string. Second is a *query suggestion*. This is a list of related terms, which you can see when typing a query into the search engine. This list may contain queries which start with what user already typed and sometimes it may also contain some spelling correction.

Traditionally, a spelling correction is employed in many aspects of human - computer interaction. It can be found for example in web browsers when filling forms or more often in a text processing application like MS Word. These are commonly a *dictionary based* correctors. For a given word, which is not in dictionary, a user is warned and the software suggests to pickup a correction from a candidates list. This list contains words from a hand-crafted dictionary which are in some way *similar* to original word.

This approach cannot be directly applied on the problem of correction of users queries in the search engine or any other similar system with text input for several reasons. First, an idea of a trusted dictionary, which will indicate correct query fails quickly, since the Internet contains a lot of information, which is growing fast. Users search for new terms every day and it would be nearly impossible to maintain such a list. For example suppose a query `bok`. Is it a misspell of the word `book` or `boc` which stands for *Bank of Ceylon*? Is it an abbreviation of a product? If it is a misspell, what if a music band with that name wins a talent contest show and becomes popular overnight? Will it still be a misspell? How do we decide?

It is pretty obvious, that some context of the search term may be useful. Suppose queries `motocykl jawa` and `motocykl java`. Both queries contain the same words, both words are *valid*, both have many references on the Internet but only one of the above queries makes a plausible sense. For example `motocykl jawa` suggests that user searches for a motorcycle of a famous brand `Jawa` and other one it is a misspell. On the other hand, it is possible that someone was searching for a motorcycle rental service in Jakarta, *Jáva* island and just don't use accents properly (some people prefer not to). In that case, it would be valid search term.

Moreover, queries contain only a little context in comparison to text processing applications, since average number of words in the search term is only about 2-3. This amount of context combined with the fact, that queries are typically not properly structured sentences, disqualifies usage of grammar checker, which we can sometimes see in the text processing applications. Thus, a statistical approach is preferred to be adapted.

Above mentioned problems illustrate the enormous complexity and difficulty of this a task. A query spelling correction is well established [18] [2], challenging [16] [7] research task formulated as follows. Given a query string q find its *correct representation* $\hat{c}$ such that a user *intended to type* $\hat{c}$. Notice that $\hat{c} = q$ is also in some cases a valid option.

The question is what does exactly mean *intended to type*. Further, we will assume that the most probable explanation corrects the error. That means that *fewer* errors are *more probable* than *more* errors. With this in mind and with the assumption that most of the users are searching for the same things, we can decide *when* the query is misspelled and *how* to correct it.

An evaluation of an algorithm solving this problem is usually done by comparison of an algorithm output with human-ranked results where more than one human must agree [7] [12] on the same answer.

We teamed with a major Czech search company *Seznam.cz* for this problem. We have built a unique query corrector system for their engine. The corrector described in this thesis focuses on the Czech search terms, uses new combination of signals to form a feature vector and applies the learning to rank technology.

The rest of the thesis is structured as follows – first we present an overview of the past development and current research in query correction task. Then we explore basic probabilistic framework called *noisy channel*. The next chapter overviews how to *model a natural language* and then we show construction of an *error model*. In addition, we examine phenomena occurring in query spelling errors. In chapter 6 we quickly introduce a *learning to rank* machine learning task, which we will use in extending noisy channel model to a *ranker based model*. Then we describe additional components in the implemented system pipeline and give performance evaluation and comparison to a baseline method. Finally we will conclude with quantitatively describing the achieved results.

2 Prior work

The oldest known system for spelling correction was proposed in 1964 by Damerau [9]. There is a short description. Let Σ be a finite-length alphabet and suppose, that $D \subset \Sigma^*$ is the clean dictionary of the words of the language. For given word $q \notin D$ find correction $\hat{c}$, such that

$$\hat{c} = \arg\min_{c \in D} dist(q, c) \tag{2.1}$$

where $dist(q, c)$ is any function $f : \Sigma^* \times \Sigma^* \to \mathbb{R}$. Levenshtein and Damerau proposed *edit distance*. The edit distance function $ed(q, c)$ of two strings $q, c \in \Sigma^*$ is the minimum number of *edit operations* which are required to make strings q and c equal. These edit operations are

- insertion of a character $\alpha \in \Sigma$ at any position in the given string
- deletion of a character at any position in the given string
- substitution of a character $\alpha \in \Sigma$ to a character $\beta \in \Sigma$ at any position in the given string
- transposition of two consecutive characters in the given string

Example: Levenshtein-Damerau distance

$$\begin{aligned} ed(\texttt{leveinstain}, \texttt{levenshtein}) &= 3, \quad \texttt{i} \to \Lambda, \Lambda \to \texttt{h}, \texttt{a} \to \texttt{e} \\ ed(\texttt{roach}, \texttt{approach}) &= 3, \quad \Lambda \to \texttt{a}, \Lambda \to \texttt{p}, \Lambda \to \texttt{p} \\ ed(\texttt{obivous}, \texttt{obvious}) &= 1, \quad \texttt{iv} \to \texttt{vi} \end{aligned}$$

The major issue of the mentioned approach is that it ignores the frequency of strings' appearance in the language. One could imagine that for the misspelled word q there will be many strings $c \in D$ with edit distance 1, with a greatly varying frequency of occurrence. In such case, there is no reason to prefer a less common word over the more commons ones. The obvious solution is incorporating information about the word occurrence in the language. Let us define slightly modified approach.

Let $q \in \Sigma^*$ and $q \notin D$ be input query find correction $\hat{c}$ such that

$$\begin{aligned} \hat{c} &= \arg\max_{c \in D} P(c) \\ \text{subject to} \quad & dist(q, c) \leq \delta \end{aligned} \tag{2.2}$$

where $P(c)$ denotes the probability of string c occurrence in the language and $\delta \in \mathbb{R}$ is the chosen upper bound on maximal allowed distance between q and c.

Formulation (2.2) solves some of the problems of the previous model - it gives us instruction how to pick up a correction when multiple words within the same distance are

available and moreover, it allows us to pick up more probable correction with (slightly) higher distance over the less probable correction with lower distance when it is convenient.

All previous models suffer from the fact, that they are *word based*. For any string q consisting of words $q_1, q_2, \ldots, q_n$, during correction we treat every single word in isolation. This can be resolved by spawning *content sensitive spelling correction* (CSSC) problem [24].

Let $s \in \Sigma^*, s = s_l q s_r$ be a query string contains a context (other words) such that $s_l, s_r \in D$ and $q \notin D$. Find suitable correction for q, such that

$$\begin{aligned} \hat{c} &= \arg\max_{c \in D} P(c|s_l q s_r) \\ \text{subject to} \quad & dist(q, c) \leq \delta \end{aligned} \tag{2.3}$$

This approach can deal with correction of multi-words queries better than previous ones. Consider following task. User types a phrase `cae parking`. In previous models the correction would be the most probable word within the distance δ. Suppose that the word `cat` is more probable than `car`. Without considering context of the misspelled word the query corrector would suggest a ridiculous correction `cat parking`. When the probability of the occurrence of the word is conditioned by its context, we could assume that suggested correction would be more reasonable.

Dictionary based spelling correction algorithms share major flaw – building and maintaining large-scale refined dictionary is very expensive and nearly impossible for web queries correction. The Internet is quickly changing and so the information which users search for. Rather then keeping up-to-date large-scale refined dictionary it is better [26] to create a specific *candidate set* for each query. This can be done by for example performing a fuzzy string match over strings extracted from the web (since it is a very large corpora of searched terms) and unionising them with all the string within certain edit distance from a query.

$$C_q = \{c \in \tilde{D} \mid sim(c, q) \geq \tau\} \ \cup \ \{c \in \Sigma^* \mid ed(q, c) \leq \delta\} \tag{2.4}$$

where $\tilde{D}$ is large, misspell containing (aka *noisy*) words dictionary compiled from large source of data (e.g. from websites on the Internet), $sim(c, q)$ is some *similarity measure* between two strings, τ is lower bound on their similarity and δ is value chosen in respect to computation limits – in real life it is crucial to get correction as soon as possible. Higher bounds on δ leads to enormous size of candidate set. Notice that in the models below we dropped the assumption, that word/query to be corrected can't be in the dictionary.

The question remains, how to choose value of δ. When it is too small, no correction could be found since the task (2.2) may become infeasible for some strings q. When it is too big, the constraint in (2.2) will play only minor role and the query corrector returns just the most probable word in the language.

One possible solution is to incorporate optimization constraint into its objective function and select correction $\hat{c}$ which is a compromise between high probable words and the closest

words (in terms of used distance metric). This fits into the well studied model in the Information Theory called *Noisy Channel* [10]. For the given string $q \in \Sigma^*$ find correction $\hat{c}$ according to

$$\hat{c} = \arg\max_{c \in C} P(q|c)P(c) \tag{2.5}$$

where $P(c)$ is probability of occurrence c in language and $P(q|c)$ is conditional probability of typing query q instead of c. Notice that $P(c)$ is not a probability (frequency) of a single word but rather the whole string c. When string c can be *tokenized*, we can use a *n-gram language model* to compute such probability. (see chapter 4).

The motivations behind formulation of (2.5) are in more details described in chapter 3. In Information Theory, the term $P(c)$ is called the *source model* and $P(q|c)$ is called the *error model* or the *channel model*. As we already mentioned, the $P(c)$ is the probability distribution of a string occurrence in a language. The $P(q|c)$ captures event, how likely it is that the user wanting to type string c but types q instead. The most simple error model is modified *edit distance* such that

$$logP(q|c) = -ed(q, c)$$

As you can see the model assigns lower probabilities for the terms with greater edit distance, which is desired.

There are many different error models - [10] used error model based on generic string-to-string edits operations, [16] examined error model based on distributional similarity measures, [24] used phonetic error model, [14] proposed a phrase-based error model and many others were studied.

In a real life implementation of the noisy channel based spelling corrector we needed relatively scale the error model score to a language model score to achieve good performance. When error model scores misspells too heavily in respect to language score, no correction is found and when it is too benevolent, corrections are usually far apart from intended term.

One solution is to find optimal weight $\lambda \in \langle 0, 1 \rangle$ for convex combination of error model and language model component such that model

$$\hat{c}_i = \arg\max_{c_i \in C} (1 - \lambda) \log P(q_i|c_i) + \lambda \log P(c_i) \tag{2.6}$$

achieves maximal *acccuracy* on human-labeled set $T = \{q_i, \hat{c}_i\}_{i=1}^m$. Essentially, the same model with additional components in a system pipeline was used in 2009 in Google Search [26].

Strictly speaking, the noisy channel framework doesn't allow incorporating nothing more then mentioned probability of string occurrence $P(c)$ and error model $P(q|c)$. One can imagine, that incorporating features like phonetic similarity of terms q and c or the number of retrieved relevant document to candidate c could be quite useful.

Even higher accuracy can be achieved [13] when task of correcting query q is transformed into *ranking* task.

For given query q find correction $\hat{c}$ such that

$$\hat{c} = \arg\max_{c \in C} h(\phi(q, c)) \tag{2.7}$$

where $\phi : \Sigma^* \times \Sigma^* \to \mathbb{R}^n$ is mapping from query-candidate pair into its *feature vector* containing (potentially) useful signals for spelling correction and $h : \mathbb{R}^n \to \mathbb{R}$ is hypothesis function which is tuned to achieve maximal out-of-sample performance with the provided dataset $T = \{q_i, \hat{c}_i\}_{i=1}^m$. Details are described in chapter 6.

In our work we have adopted the last model, which is currently (2013) arguably considered as the state of the art [12].

3 Exploring noisy channel model

The noisy channel model is a mathematical framework for modelling communication in a noisy enviroment. Many task such as optical character recognition, speech recognition and our task of query spelling correction can be inspired by such model and we provide a basic view into this tool and later show, how to extend this model.

3.1 Definition

Noisy channel is model of point-to-point communication. It assumes that one person is sending symbols from some alphabet through communication channel to another person which is listening on the other side. Transmitter transmits symbols $c \in \mathcal{X}$ underlying probability distribution $P(c)$ referred as *source model*. On the other side of the channel receiver receives symbols $q \in \mathcal{Y}$ underlying distribution $P(q)$.

The channel is fully described by transition matrix $\mathbf{M} \in \langle 0, 1 \rangle^{|\mathcal{X}| \times |\mathcal{Y}|}$ where

$$m_{ij} = P(q_j | c_i) \tag{3.1}$$

which represents probability of event transferring symbol $c_i \in \mathcal{X}$ to $q_j \in \mathcal{Y}$ (noise).

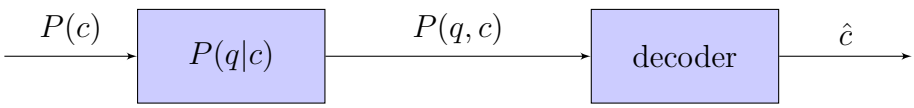


Figure 1: Noisy channel model

A decoder is any function $h : \mathcal{Y} \to \mathcal{X}$. Naturally, we want a decoder which minimizes probability of wrong guess, hence optimal decoder h^* is

$$\begin{aligned} p_e &= E_{(q,c)}[\![h(q) \neq c]\!] \\ h^* &= \arg\min_{h \in \mathcal{H}} p_e \end{aligned} \tag{3.2}$$

Where $[\![\ldots]\!]$ detones the indicator function.

There are several ways how to decode symbols (queries) sent over the channel. When we know the source distribution, one way to go is decoding symbols c in terms of maximizing joint distribution $P(q, c)$. Using Bayes' theorem we calculate

$$\hat{c} = \arg\max_c P(q, c) = \arg\max_c P(q|c)P(c) \tag{3.3}$$

also known as *MAP - maximum a posteriori* estimation.

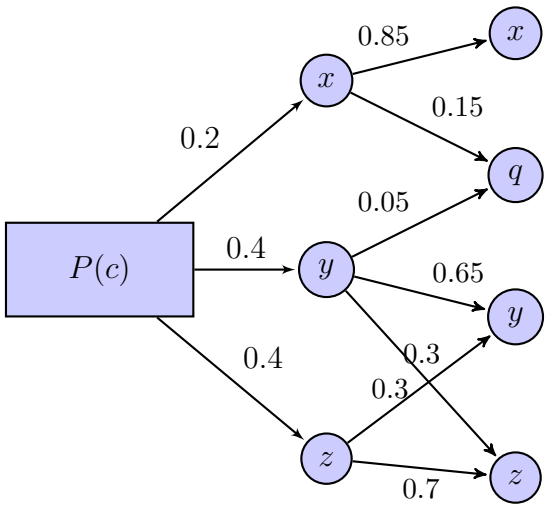


Figure 2: Sample noisy channel with known source distribution

Example: Simple channel with a MAP decoder Suppose channel with $\mathcal{X} = \{x, y, z\}$ and $\mathcal{Y} = \{q, x, y, z\}$ and known source distribution as plotted in Figure 2.

We can construct transition matrix as

$$\mathbf{M} = \begin{bmatrix} P(q|x) & P(x|x) & P(y|x) & P(z|x) \\ P(q|y) & P(x|y) & P(y|y) & P(z|y) \\ P(q|z) & P(x|z) & P(y|z) & P(z|z) \end{bmatrix} = \begin{bmatrix} 0.15 & 0.85 & 0 & 0 \\ 0.05 & 0 & 0.65 & 0.3 \\ 0 & 0 & 0.3 & 0.7 \end{bmatrix} \tag{3.4}$$

Since we know the source distribution, it is convenient to use MAP decoder. Suppose that we sent some symbol through the channel and observed letter q. The question is, what character had been sent originally.

Using equation (3.3) we compute all the possibilities and obtain

$$\begin{aligned} P(q|x)P(x) &= 0.15 \cdot 0.2 = \mathbf{0.03} \\ P(q|y)P(y) &= 0.05 \cdot 0.4 = 0.02 \\ P(q|z)P(z) &= 0 \cdot 0.4 = 0 \end{aligned}$$

so according MAP it is most likely that symbol x was sent through the channel. It seems intuitive, since the symbol z cannot be sent when output is q at all, so x and y remains. But even despite the fact that the symbol y is sent more often *a priori* then x we know, that the probability of error $x \to q$ is three times bigger than $y \to q$ so resulting guess would be x.

3.2 Understanding

In our task the *source model* $P(c)$ represents distribution of all the *valid queries*. Notice that noisy channel (and underlying MAP estimation) assumes that we know this distribution *a priori*.

In real life we don't know this distribution since we posses only a complete histogram of all queries typed into the search engine with no additional information what is valid and what is not. When we explore data in form of histograms we observe, that many terms typed only once are valid search terms and on the other hand some very popular terms are misspelled (like `gogle`) so frequently that it is impossible to set some kind of threshold of occurrence which distinguishes misspelled terms from valid ones (see Table 1).

Conclusion is that $P(c)$ distribution can be only crudely approximated by taking all queries and assume that with more data available the noise will have fewer influence on the resulting distribution. How to build such model is described in chapter 4.

Channel model $P(q|c)$ is usually estimated from query reformulation sessions [14] or click-logs [13]. Resulting models may greatly vary in complexity and performance. There are great number of them from simpler ones like *weighted edit distance* through *Brill's and Moore's substring* model to a *mixture models* employing distributional similarity measures or phonetic similarity.

4 Natural language modeling

In our task we need to model a *source distribution* for a noisy channel model mentioned above. This is done by a language model – it is a distribution over all possible strings occurring in a given language. In a natural language this distribution is enormously complex and thus difficult to build.

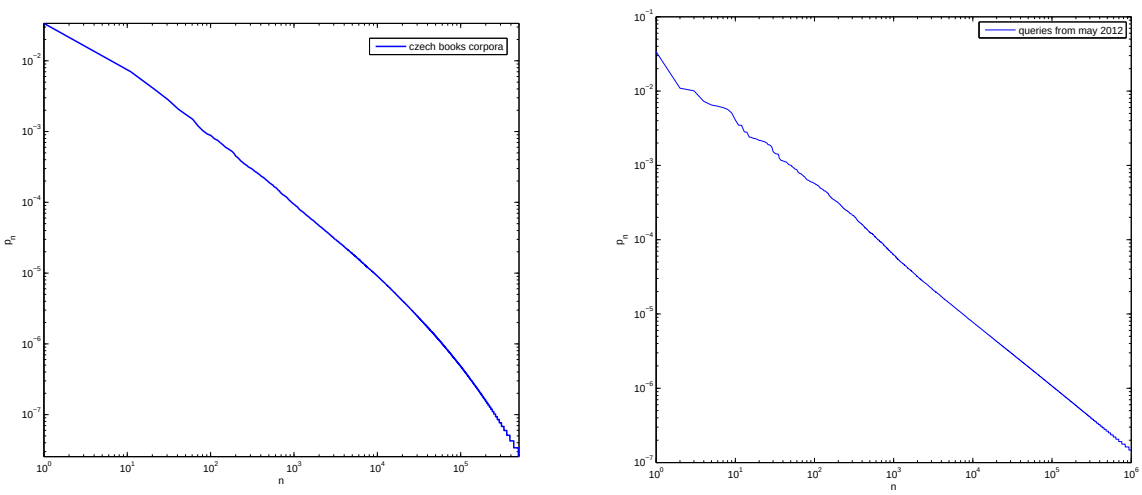
Later in this chapter we show, how we built a large model from over 500 M unique queries entered by users to the Czech biggest search engine Seznam.cz which we used in the following experiments.

4.1 Introduction

Zipf observed [28] that distribution of words in any human language tends to follow formula

$$p_n = \frac{k}{n} \tag{4.1}$$

referred as *Zipf's law*. The p_n is the probability of occurrence nth most occurring word in the large corpus of given language and k is a constant dependent on the language. Taking a logscale on both axis, resulting distribution is approximately plotted as a linear function (see Figure 3).



(a) Czech books corpora

(b) May 2012

Figure 3: Queries distribution

Such distribution is referred as *LNRE – large number of rare events* model [11]. Notice that majority of the words is occurring so rarely – about a half of queries have only single occurrence (referred as *hapax legomena*). One could think that these are misspelled words

but in fact, more then half of them are completely valid words (see Table 1). To capture (possibly) all such queries, we need a great amount of the data available, especially when estimating language models from the noisy source (which queries logs certainly are).

query	occurrence
`Vochlice - rozhledna`	1
`Vip Champagne Lounge`	1
`Uveďte příklady, kdy náklad je současně výdajem.`	1
`ustanovení § 663 a násl. zákona č. 40/1964 Sb`	1
`úprava názvu Kojeneckých ústavů v ČR`	1
`ts.leagueoflegends.cz:9988`	1
`TOURATECH Jilove foto`	1
`TOURATECH ADVENTURE DAYS & Travel Event 18-20.5.2012`	1
`TOURATECH ADVENTURE DAYS`	1
`Tobiášův vrch, rozhledna`	1

Table 1: Random sample of *hapax legomena* queries in May 2012

4.2 Models

A statistical language model is a probability distribution $P(s)$ over strings s which reflects, how likely it is that string s appears in the large corpus of text [21]. These models are widely used in task like machine translation, optical character recognition, information retrieval and many others. The most popular SLM[1] is the *n-gram* model but certainly not the only one (e.g. maximum entropy models are powerful and outperforms n-gram models) [21].

N-gram model in computational linguistics models language as $(n-1)$th Markov word model. That means that probability of occurrence of a certain word in the language is conditioned by co-occurrence of previous $n-1$ words. C. E. Shannon [22] showed, that human language when modeled as $(n-1)$th Markov model has low entropy and still produce reasonable results. Let us define such model.

Let $s = w_1, w_2, \ldots, w_k$ be a sentence. Probability of occurrence of such sentence in large corpus of text can be computed as

$$P(s) = P(w_1, w_2, \ldots, w_k) = \prod_{i=1}^{k} P(w_i|h_i) \tag{4.2}$$

where the term $h_i = \{w_1, \ldots, w_{i-1}\}$ is referred as *history*. Under a *n-gram model* we assume

[1] statistical language model

$$P(w_i|h_i) \approx P(w_i|w_{i-n+1}, \ldots, w_{i-1}) \tag{4.3}$$

so that *history* contains only preceding $n-1$ words. Notice, that the history behind the start of the sentence is not defined since words w_i, $i \leq 0$ doesn't exist. To make definition meaningful for the first word, we use *start of the sentence* token, meaning that occurrence of the first word of the sentence is modeled as $P(w_1|\texttt{<s>})$.

Probability $P(w_i|h_i)$ is usually estimated using MLE (maximum likelihood estimation) from large source of text data as [8]

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{\#(w_{i-n+1}^{i})}{\sum_{w_i} \#(w_{i-n+1}^{i})} \tag{4.4}$$

where the $\#(w_{i-n+1}^{i})$ denotes a occurrence of the string $w_{i-n+1}, \ldots, w_{i-1}, w_i$.

Example: Bigram model Probability of a sentence is calculated as

$$P(\texttt{I like cats}) = P(\texttt{I}\,|\,\texttt{<s>})P(\texttt{like}\,|\,\texttt{I})P(\texttt{cats}\,|\,\texttt{like})P(\texttt{</s>}\,|\,\texttt{cats})$$

where used bigrams are estimated from data source as

$$\begin{aligned}
P(\texttt{I}|\texttt{<s>}) &= \frac{\#(\texttt{<s>}, \texttt{I})}{\sum_w \#(\texttt{<s>}, w)} \\
P(\texttt{like}|\texttt{I}) &= \frac{\#(\texttt{I}, \texttt{like})}{\sum_w \#(\texttt{I}, w)} \\
P(\texttt{cats}|\texttt{like}) &= \frac{\#(\texttt{like}, \texttt{cats})}{\sum_w \#(\texttt{like}, w)} \\
P(\texttt{</s>}|\texttt{cats}) &= \frac{\#(\texttt{cats}, \texttt{</s>})}{\sum_w \#(\texttt{cats}, w)}
\end{aligned}$$

N-grams models are simple and relatively successful in many task. Notice that this model doesn't use any extra information about objects it models (language). No assumption about structure, form or any other aspect of natural language aren't used.

Downside is that large volume of data is neccessary to estimate such models. Even simple bigram model for English requires several hundred millions words to fully saturate [21]. On the other side, different SLM (like maximum entropy models [3]) achieves lower perplexity with the same training data available, but takes much longer time to train and evaluate.

Estimating probabilities using MLE over a sparse data may be tricky. Suppose we want to compute a probability of

$$P(\texttt{A slime eats cat})$$

when we could face the problem, that the occurrence probability of a word `cat` with history containing words like `slime` and `eats` would be very low or even zero, becouse it has not been seen ever before. Maximum likelihood principe tells us to assign zero probability to events, which have not occurred in the data at all.

So in that case resulting probability of such sentence under mentioned model would be zero, which is undesirable since even when sentence `A slime eats cat` sounds odd, one would expected, that it is still more probable then random strings like `fsdaf axdqw mboi qql`. A method developed to deal with such cases is called *model smoothing*.

4.2.1 Smoothing techniques

The general idea of smoothing is to assign non-zero probability to events which we never saw before and/or adjust other low probability events. The simplest method which can be used [8] is referred as *δ-additive smoothing*

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{\delta + \#(w_{i-n+1}^{i})}{\delta|V| + \sum_{w_i} \#(w_{i-n+1}^{i})} \tag{4.5}$$

which simply adds δ to the total count of occurrence and $|V|$ is the number of unique words in the model. Since $\delta > 0$ model assigns non-zero probability to all events.

Other method is *Good-Turing frequency estimation. Good-Turing* estimation redistributes part of probability of the low probability events to the zero-prob event. It adjust occurrence of every n-gram, that has occurred r times by counting it $\hat{r}$ times instead [8]. Do it so by

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{\hat{r}}{\sum_{r=1}^{\infty} r n_r} \tag{4.6}$$

$$\hat{r} = (r+1)\frac{n_{r+1}}{n_r} \tag{4.7}$$

where n_r is a number of n-grams that has occurred r times. Good-Turing smoothing is a popular smoothing method in many task, where some sort of frequency estimation is needed.

Every our language model described below uses Good-Turing smoothing, although more smoothing methods (especially for language modeling) exists [8]. We don't provide more specific details since the implementation of smoothing techniques isn't part of the thesis.

4.3 A Web-scale n-grams models

4.3.1 Data source

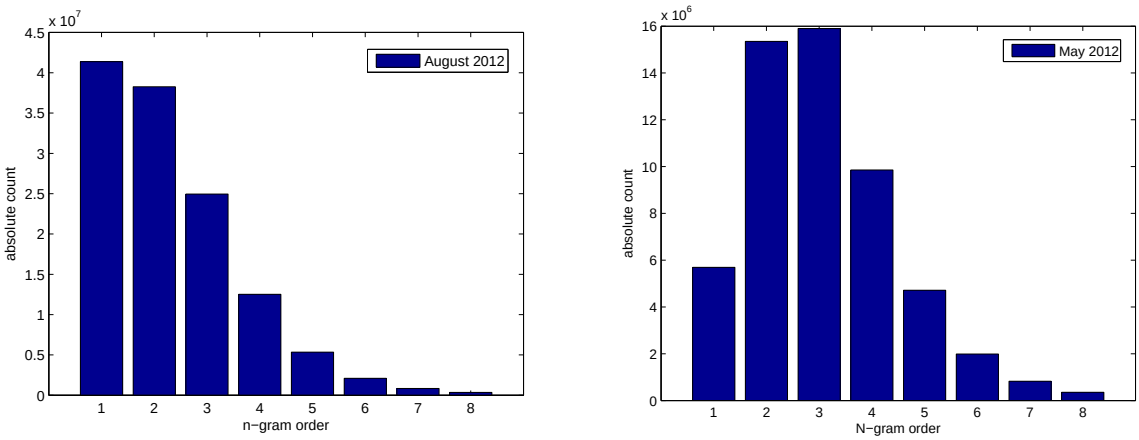
We have collected query logs histogram from 11 months of traffic on the major Czech search engine (Seznam.cz). Histograms takes approximately 15 GB of disk storage and consists of more then 500 M unique queries.

Aside of that we have extracted all Czech Wikipedia articles and have build a 4-gram language model of it. Resulting dataset is much smaller, taking only approximately 500 MB of disk storage uncompressed.

Performance of these models in terms of perplexity are presented below (see chapter 4.4.3).

4.3.2 Query analysis

We have observed that n-gram spectra of the queries greatly vary over the months - see Figure 4.



(a) August 2012

(b) May 2012

Figure 4: N-gram order of queries

Since the amount of 4-gram queries is significant, we assume that 4-gram model outperforms 3-gram (see chapter 4.4.3). On the other hand, we don't expect that building 5-gram model would be such benefit (morover not enough data for such large model is available).

4.4 Implementation and performance

4.4.1 SRILM

Instead of implementing powerful language model system we decided to use state of the art language modeling toolkit *SRILM*[2] developed since 1995 at SRI Speech Technology and Research lab[3]. Over the last 15 years they have implemented [23] a system for serving

[2]http://www.speech.sri.com/projects/srilm/download.html
[3]http://www.speech.sri.com/

statistical language models capable to handle very large-scale models (tens of gigabytes in memory) and whole toolkit for building such models from data.

SRILM toolkit provides tools for data aggregation, language model training (n-gram model, maximum entropy model, mixture model etc.) and various smoothing techniques. Other functionality is serving such models to applications. Since it uses (in memory) effective data structures it provides extremely fast (see measurement in section 4.4.2) n-gram service with reasonable memory consumption (our biggest model *11m@4-gram+queries* takes approx. 13 GB in memory on 64bit architecture).

SRILM comes in form of C++ source code which has to be compiled before using. With additional changes in SWIG binding[4] we managed to ask LM from Python (similar approach as described in [17]).

Disadvantage of SRILM (in 2013) is that it can't run distributed on many machines (like implemented system in [14] referred as *distributed n-gram LM platform*), so the model size and its performance is caped by limit of a single machine.

4.4.2 N-gram retrieval

We measured n-gram retrieval performance of SRILM language modeling toolkit when asking from Python. Model used for evaluation contained 58 M 1-grams, 17 M 2-grams and 16 M 3-grams. Results are plotted in Figure 5 with 95% confident intervals.

Results were obtained by taking the minimum of three runs. Peaks on the graphs are caused by hash collisions on certain queries.

model	median [μs]	95% conf. int. [μs]
1-gram	8.10	$\langle 7.86, 10.01 \rangle$
2-gram	15.02	$\langle 10.96, 17.16 \rangle$
3-gram	20.02	$\langle 15.97, 23.12 \rangle$

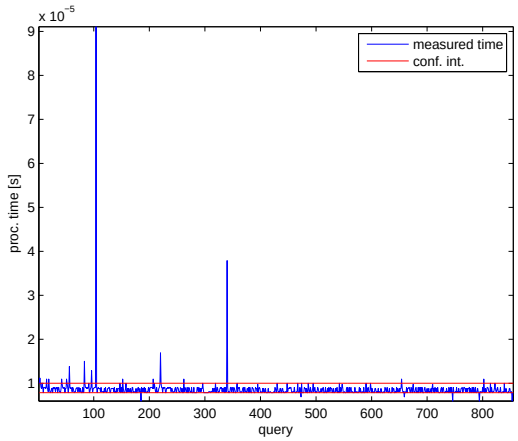
Table 2: SRILM retrieval performance

Tests were performed on the commodity hardware on the Linux 64-bit platform. As expected, unigram retrieval is faster then higher order n-grams. We also observed higher variance in higher order n-gram.

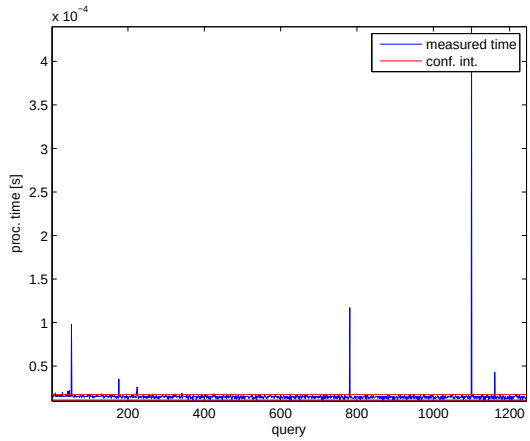
4.4.3 Perplexity

Perplexity is the function of both model and the data. When viewed as a function of the data (language) only, it estimates entropy of the data. When it's considered as a function

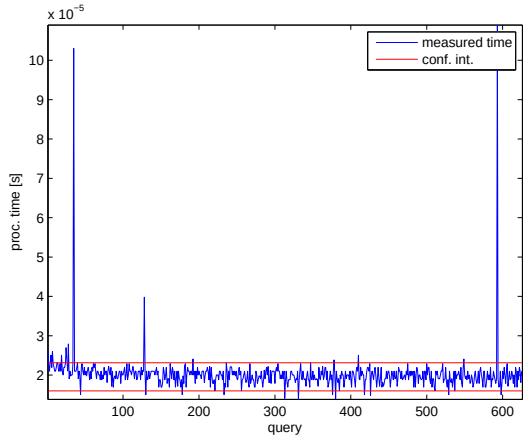
[4] https://github.com/jc/languagemodel



(a) 1-gram



(b) 2-gram



(c) 3-gram

Figure 5: N-gram retrieval with 95% conf. interval

of the model, it measures how good the model is in predicting the data. The lower results mean that the model is less uncertain about given data, meaning the lower perplexity is better.

Perplexity of the model q can be computed as

$$\text{perplexity} = 10^{H(\tilde{p},q)} \in \langle 1, +\infty) \tag{4.8}$$

where $H(\tilde{p}, q)$ is the *cross entropy* calculated as

$$H(\tilde{p}, q) = -\sum_{x \in \text{event in test sample}} \tilde{p}(x) \log q(x) \tag{4.9}$$

where $\tilde{p}$ is the empirical distribution of the test data and q is the model. Perplexity is averaged over the number of the words in the test data.

Example: perplexity of a perfect model Imagine an unigram model, which is build for a language $\mathcal{L}$ containg the only word `whiskey` (no smoothing included), so the resulting model is

$$q(x) = \begin{cases} 1 & x = \texttt{whiskey} \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

Since language $\mathcal{L} = \{\texttt{whiskey}\}$, the test sample contains only this word. When evaluating perplexity of such model on the data from language $\mathcal{L}$ we obtain

$$\text{perpexity} = 10^{H(\tilde{p},q)} = 10^{-\tilde{p}(\texttt{whiskey})\log q(\texttt{whiskey})} = 10^{-\log 1} = 10^0 = 1$$

So the model is perfect in predicting language $\mathcal{L}$.

We have evaluated different language models on the same data - search queries from a search engine. One group consists of language models based on search logs. There we examined influence of data size (11 vs. 8 vs. 6 months of queries logs traffic) and order of n-gram model used (1-gram to 4-gram order with 11 month data). On the other hand, as a comparison of an influence of the data source we built 4-gram model from Czech Wikipedia articles (*4-gram+cs-wiki-body*).

language model	perplexity
11m@1-gram+queries	9301
11m@2-gram+queries	1386
11m@3-gram+queries	825
11m@4-gram+queries	**701**
8m@4-gram+queries	803
6m@4-gram+queries	880
4-gram+cs-wiki-body	11121

Table 3: Perplexity results on test queries

We have shown (as [14] too), that models based on the queries logs achieve much lower perplexity than other ones. Of course, the more data we have, the resulting model is better. The question is, how is the perplexity of the language model related to the accuracy in query spelling correction task. Lower perplexity doesn't necessarily imply better performance in all tasks where language model is employed, but usually decrease of perplexity more then 10-20% is considered as significant [21]. Results of models discussed above are summarized in the Table 3.

As we can see, it is crucial to build a model from queries logs, since they achieve significantly lower perplexity on the target data for query spelling correction task. We also observed, that usage of 4-gram model instead of common choice 3-gram [10] [21] gains noticeable benefitin terms of perplexity.

5 Error model

The task of the error model is to determine, how likely it is, that the user types a query q when he/she meant to type c instead. This event can be captured by probability distribution $P(q|c)$ over all strings. Such distribution would be enormously complex, thus some simplification are made.

For training models mentioned below we extracted 185 M (q_i, q_{i+1}) pairs from traffic of the Czech major search engine. Resulting pairs are created from what we refer as *query reformulation session*. We identified a single search session (according IP address, browser, etc.) and logged (anonymized) pairs of queries which were typed into search engine consecutively and were maximally 4 edit operation apart.

q_i	q_{i+1}
`pil`	`pila`
`poštovní z ameriky`	`poštovné z ameriky`
`magnaticky vlorky`	`magneticky vložky`
`noje noty`	`moje noty`
`zaklinac 2 oblehani vergenu`	`zaklinac 2 oblehani bergenu`
`zaklinac 2 oblehani vergenu`	`zaklinac 2 oblehani wergenu`
`dostat se ki owerth`	`dostat se k iowerth`
`omalovánky prasátko pepina`	`omalovánky prasátko pigy`

Table 4: An example from anonymized reformulation sessions

Unfortunately, we don't track clicks-through in search session, so data are possibly noisy. E.g. common pattern is that the user is searching for a furniture store. For example the user types `ikea` and then changes his mind and searches for `kika` (which is just 2 operation apart). Both of these words are correct names for furniture stores, but doesn't represent a correction of previous search term.

Let us derive some commonly used models.

5.1 Fixed cost edit distance

Under this model we penalise all errors in the same way. Under the assumption, that it is more likely that users make fewer errors over the more errors in typed word, we state that

$$\log P(q|c) = -ed(q, c) \tag{5.1}$$

where $ed(q, c)$ states for edit distance of strings q and c.

Weaknesses of this models are obvious – one would expected (and would be right) that some type of errors are more probable than the others. Since this trend can't be captured by fixed cost edit distance model, other were proposed.

5.2 Weighted edit distance

Let e be a *transfeme* - event that a character $\alpha \in \Sigma$ (or a whole group of characters) is changed into $\beta \in \Sigma$ during typing. For example it could be an event that character `y` is misspelled as `i` (`y` $\rightarrow$ `i`) or that character `o` is missing (`o` $\rightarrow \Lambda$).

Under the assumption that occurrence of error in word (query) is independent on other errors we can write probability of misspelling word c as

$$P(q|c) = \prod_{i=1}^{n} P(e_i) = P(e_1) \cdot P(e_2) \cdots P(e_n) \tag{5.2}$$

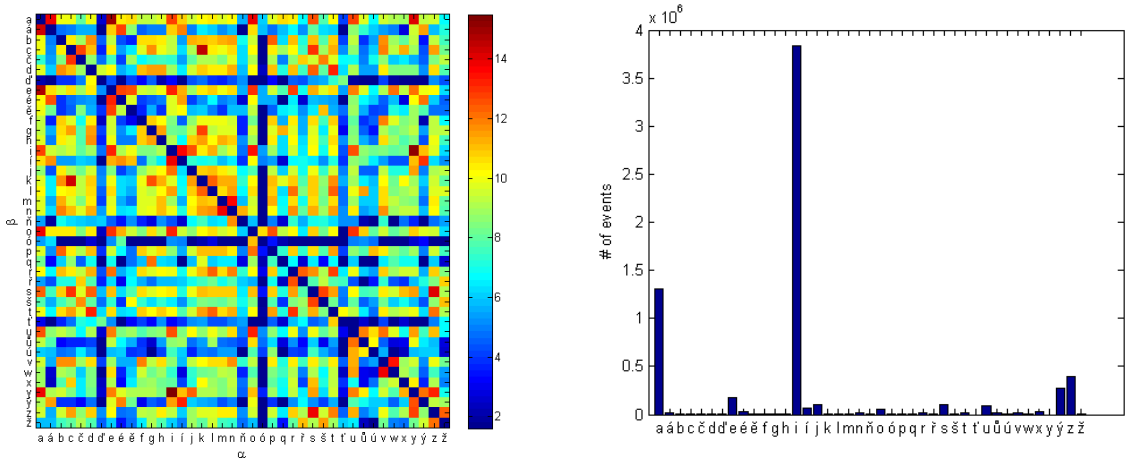
Taking the logarithm of the expression (5.2) we obtain

$$\log P(q|c) = \log P(e_1) + \log P(e_2) + \ldots + \log P(e_n) \tag{5.3}$$

So one can see, that our method of penalisating errors is additive with respect to number of errors that occurred. This suggests us an idea to make

$$ed_w(q, c) \propto -\log P(q|c) \tag{5.4}$$

Model $ed_w(q, c)$ is called weighted edit distance, since it does not treat all errors equally but rather considers probability of each transfeme individually. By analysis of query logs reformulation sessions we observed, that probability of transfemes is greatly varying (see Figure 6.



(a) Misspell map, logscale heatmap

(b) Distribution of misspells of letter y

Figure 6: Misspell map

One can notice of fact that misspell map is symmetric - that mean that probability of transfeme $\alpha \rightarrow \beta$ is about the same as $\beta \rightarrow \alpha$. Moreover, we identified two major source of misspelling - cognitive/grammar errors and keyboard layout errors.

Cognitive errors are caused by absence of knowing of correct spelling - for example in Czech one of the most common mistake is usage of incorrect letter `y` and `i`. As one can see, distribution $\alpha \rightarrow$ `y` in Figure 6 is greatly biased toward to `i`.

On the same figure one can observe some portion of occurrence of transfeme `z` $\rightarrow$ `y` which is caused by other source of errors - the keyboard layout (QWERTY vs. QWERTZ). Another example is heatspot in Figure 6 in area of keys `ljkmn` which are very close on the keyboard, so possible mistyping is potential threat.

Probability $P(q|c)$ in this model can be computed simply by dynamical programming algorithm (see Algorithm 1) with computational complexity $\mathcal{O}(|q| \cdot |c|)$.

Algorithm 1: Weighted edit distance

input : strings c and q
output: $\log P(q|c)$

```
F_{0,0} ← 0
for i = 1 to |c| do
    F_{i,0} ← − log P(Λ|c_i)
end
for j = 1 to |q| do
    F_{0,j} ← − log P(q_j|Λ)
end
for i = 1 to |c| do
    for j = 1 to |q| do
        F_{i,j} ← min{F_{i−1,j} − log P(Λ|c_i), F_{i,j−1} − log P(q_i|Λ), F_{i−1,j−1} − log P(q_i|c_i)}
    end
end
return −F_{|c|,|q|}
```

The main issue of this model is that we want to model e.g. $P(\texttt{facebok}|\texttt{facebook})$ as $P(\texttt{oo} \rightarrow \texttt{o})$ rather then $P(o \rightarrow \Lambda)$ since this is what actually happened. User didn't forgot to type a single `o`, but in fact, typed `o` instead of `oo`.

We could resolve that by considering the Markov assumption, which states that every transfeme $\alpha \rightarrow \beta$ is conditioned by previous character(s). Brill and Moore proposed [10] much more general model.

5.3 Brill's & Moore's error model

The probability of typing query q instead of c is modeled [10] as

$$P(q|c) \approx \max_{R \in Part(c), T \in Part(q)} \prod_{i=1}^{|R|} P(T_i|R_i) \quad \text{subject to} \quad |R| = |T| \tag{5.5}$$

where $Part(x)$ denotes the set of all possible partitioning string x with segments of maximum length L.

For example, let $L = 2$ and $x =$ `book`, then $Part(x)$ contains alignments like (b, o, o, k), $(bo, o, k), (b, oo, k), (bo, ok)$ and so on. Optimization constraint makes sure that R and T is one-to-one segment alignment and formula (5.5) can be computed.

Probabilities of transfemes are estimated from query reformulation sessions in following way. Two consecutive queries q_i and q_{i+1} are aligned based on their edit distance. Then, we generate all possible edit operations up to segment length L. Suppose a query pair (`telefhone`, `telephone`), then the generated transfemes looks like

- $L = 1$: `p` $\rightarrow$ `f`
- $L = 2$: `p` $\rightarrow$ `f`, `ph` $\rightarrow$ `fh`, `ep` $\rightarrow$ `ef`
- $L = 3$: `p` $\rightarrow$ `f`, `ph` $\rightarrow$ `fh`, `ep` $\rightarrow$ `ef`, `lep` $\rightarrow$ `lef`, `pho` $\rightarrow$ `fho`, `eph` $\rightarrow$ `efh`

These transfemes are generated from 20 GB worth query reformulation sessions using map-reduce algorithm as described below. The process took a small tens of CPU hours.

Example: notorious error We wish to compute $P(\texttt{facebok}|\texttt{facebook})$. Since estimated probability of `oo` $\rightarrow$ `o` is bigger then `o`$\rightarrow \Lambda$, the event `facebook`$\rightarrow$`facebok` is modeled by

$$P(\texttt{facebok}|\texttt{facebook}) = P(\texttt{f}|\texttt{f})P(\texttt{a}|\texttt{a})P(\texttt{c}|\texttt{c})P(\texttt{e}|\texttt{e})P(\texttt{b}|\texttt{b})P(\texttt{o}|\texttt{oo})P(\texttt{k}|\texttt{k}) \tag{5.6}$$

Generating the set of all partitioning has exponential computational complexity. Fortunately, the standard dynamic programming algorithm for filling weighted distance matrix can be applied with minor changes and computational complexity drops to $\mathcal{O}(|c|^2 \cdot |q|^2)$ (see Algorithm 2).

5.4 Extended Brill's & Moore's model

Previously mentioned model can be extended by additional considering position of the errors. One would expect, that distribution of error would be different on the start of the word (prefixes) from ending (suffixes).

Suppose that we distinguish $POS = \{start, middle, end\}$ positions of errors in query / candidate pair. The task is to model distribution of transfemes conditioned by position of their occurrence

$$P(\alpha \rightarrow \beta \,|\, POS) \tag{5.7}$$

The rest of the algorithm remains the same. [10] showed, that positional information significantly improves accuracy.

Algorithm 2: Brill's & Moore's cost align

input : strings c and q
output: $\log P(q|c)$

$F_{0,0} \leftarrow 0$
for $i = 1$ *to* $|c|$ **do**
  $F_{i,0} \leftarrow -\log P(\Lambda|c_i)$
end
for $j = 1$ *to* $|q|$ **do**
  $F_{0,j} \leftarrow -\log P(q_j|\Lambda)$
end
for $i = 1$ *to* $|c|$ **do**
  for $j = 1$ *to* $|q|$ **do**
    $F_{i,j} \leftarrow \min\{F_{i-1,j} - \log P(\Lambda|c_i),\ F_{i,j-1} - \log P(q_i|\Lambda),\ F_{i-1,j-1} - \log P(q_i|c_i)\}$
      `// in addition, we take min. prob. of transfemes in the square of length` L
    $F_{i,j} \leftarrow \min\{F_{(i-L)\ldots i,(j-L)\ldots j} - \log P(q_{(i-L)\ldots i}, c_{(j-L)\ldots j}\}$
  end
end
return $-F_{|c|,|q|}$

5.5 Other models

There many other notice-worth models - [16] proposed error model based on distributional similarity of the context of corrected word. The basic idea is, that misspelled words shares the same context as its correct form. On the other hand, words with similar spelling but with different meaning would appear in different context and model wouldn't try to correct one to another.

We tried to implement this model, but it was only partly functional since we didn't have enough misspelled words with rich context in query logs.

Other promising model [24] combined a statistical based phonetic model and previously proposed Brill's and Moore's substring model which achieved error reduction of 23.8%.

6 Learning to rank

6.1 Introduction

Learning to rank is supervised machine learning task for training models capable of ranking (ordering) things. By things we refer elements of an arbitrary set – it could be characters, letters, strings, documents, vectors or any mathematically describable objects.

This task has broad usage in Information Retrieval, Machine Translation, Natural Language Processing and others areas [15]. We have used ranking models to select top-ranked candidate as a correction to given query.

6.2 Evaluation metrics

Quality of an ordering is commonly measured by an information retrieval measure *NDCG* - Normalized Discounted Cumulative Gain. Suppose that we wish to know quality of ordering $\pi_i = (c_{i,1}, c_{i,2}, \ldots, c_{i,n})$ with associated grades $\mathbf{y}_i = (y_{i,1}, y_{i,2}, \ldots, y_{i,n}) \in \mathcal{Y}^n$ where grades $y_{i,j}$ can be for example degree relevance of $c_{i,j}$.

DCG at position k for permutation π_i of candidates is defined as

$$DCG(k) = \sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}} - 1}{\log_2(1 + \pi_i(j))} \tag{6.1}$$

where $\pi_i(j)$ is a position of candidate $c_{i,j}$ in permutation π_i of $\mathcal{C}_i$ and $\mathbf{y}_i$ is a vector of grades of candidates in ordering π_i. Common choice is to normalize DCG, such that

$$NDCG(k) = G_{max,i}(k)^{-1} DCG(k) \tag{6.2}$$

where $G_{max,i}(k)$ is normalization factor, such that a perfect ranking π_i^* has $NDCG(k)$ has value of 1.0.

Other evaluation measure is MAP - mean average precision. Average precision is given [15] by

$$AP = \frac{\sum_{j=1}^{n_i} P(j) y_{i,j}}{\sum_{j=1}^{n_i} y_{i,j}} \tag{6.3}$$

$$P(j) = \frac{\sum_{k:\pi_i(k)\leq\pi_i(j)} y_{i,k}}{\pi_i(j)} \tag{6.4}$$

When speaking in terms of precision, grades have to be neither 0 or 1, so some threshold of sufficient grade is chosen which are considered as 1's. Finally, MAP is mean across all the queries in the sample.

These are generally used evaluation metrics for quality of ordering. In our task we care mostly about precision at position 1, since search engine (typically) doesn't offer more suggestions to correction.

Example: quality of an ordering Suppose we have permutation of candidates $\pi_1 = \{\texttt{toilet}, \texttt{toylet}, \texttt{toilete}\}$ with associated grades for candidates $\mathbf{y}_1 = \{5, 0, 2\}$. Quality of ordering π_1 in terms of above mentioned metrics are

$$\begin{aligned} NDCG_1(3) &= 0.9881 \\ MAP_1(3) &= 0.8333 \\ P_1(1) &= 1.0 \end{aligned}$$

Now suppose that we swap first two candidates, so the new ordering is $\pi_2 = \{\texttt{toylet}, \texttt{toilet}, \texttt{toilete}\}$, $\mathbf{y}_2 = \{0, 5, 2\}$ and evaluations are

$$\begin{aligned} NDCG_2(3) &= 0.6402 \\ MAP_2(3) &= 0.3889 \\ P_2(1) &= 0.0 \end{aligned}$$

6.3 Learning to rank applied to query candidates ordering

Below we present a formulation of learning to rank task to a problem concerning candidate ranking for query spelling correction.

Suppose that $\mathcal{Q} = \{q_1, q_2, \ldots, q_m\}$ is the query set and $\mathcal{C}_i = c_{i,1}, c_{i,2}, \ldots, c_{i,n}$ is the candidate set for given query q_i. Let $\mathcal{Y} = \{+1, 0\}$ be the label set and suppose that for each query q_i we have a vector of labels $y_i \in \mathcal{Y}^{|\mathcal{C}_i|}$ where the j-th element of y_i says whether is $c_{i,j}$ acceptable as the candidate for correction (+1) or not (0) for query q_i.

Further suppose that we have the map $\phi : (q_i, c_{i,j}) \to \mathbb{R}^n$ which maps query - candidate pair to a *feature vector* $\mathbf{x}_{i,j}$. A feature vector is a numeric vector containing entries called *features* (signals) - these are (usually binary or real valued) entries, which contains some information about query/candidate pair $(q_i, c_{i,j})$.

Learning to rank is a task of learning hypothesis function h that represents a total order on a set of features vectors. Partial ordering on the set is defined as a relation between two elements within the set which is antisymmetric, transitive and reflexive. A total ordering or *linear ordering* is the ordering, where every two elements within the set are comparable. Strictly speaking, for our task the resulting hypothesis doesn't have to be an order relation necessary, but it's sufficient to be a *quasi-order*.

Ordering of candidates $\mathcal{C}_i$ for query q_i is then given by descent sorting of theirs scores. More precisely, the h represents the total order on the every set of candidates $\mathcal{C}_i$ such that

$$\forall q_i \in \mathcal{Q} \quad \forall c_{i,j}, c_{i,k} \in \mathcal{C}_i : \quad c_{i,j} \preceq c_{i,k} \quad \Longleftrightarrow \quad h(\phi(q_i, c_{i,j})) \leq h(\phi(q_i, c_{i,k})) \tag{6.5}$$

In order to find hypothesis h, we cast following supervised *learning to rank* task [15]. For given training set $T = \{q_i, \mathcal{C}_i\}_{i=1}^m$ find hypothesis function $h \in \mathcal{H}$ such that an evaluation metrics (mentioned above) is maximized on the out-of-sample points.

In our application to the search query correction we consider hypothesis functions to be linear in form of $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. We use following setup. Suppose relation R defined as

$$\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^n : \quad \mathbf{a}R\mathbf{b} \iff \mathbf{w}^T\mathbf{a} \leq \mathbf{w}^T\mathbf{b} \tag{6.6}$$

Next consider relation S defined as

$$\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^n : \quad \mathbf{a}S\mathbf{b} \iff \mathbf{a}R\mathbf{b} \wedge \mathbf{b}R\mathbf{a} \tag{6.7}$$

One can easily show, that S is equivalence relation on $\mathbb{R}^n$ (it is reflexive, symmetric and transitive), thus it generates an equivalence classes. Then, we define an order relation $\preceq$ on those equivalence classes

$$[\mathbf{a}]_S \preceq [\mathbf{b}]_S \iff \mathbf{a}R\mathbf{b} \tag{6.8}$$

Since $\preceq$ is obviously dichotomous ($\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^n : \mathbf{w}^T\mathbf{a} \leq \mathbf{w}^T\mathbf{b} \vee \mathbf{w}^T\mathbf{b} \leq \mathbf{w}^T\mathbf{a}$), every two elements within $\mathbb{R}^n$ are comparable, thus $\preceq$ is a total order on equivalence classes from $\mathbb{R}^n$.

Then, we can order candidates using $\preceq$ relation mentioned above

$$c_1 \preceq c_2 \preceq c_3 \preceq \ldots \preceq c_n \tag{6.9}$$

and select the maximum element of such ordered set as top ranked candidate.

6.4 Ranker based spelling correction

A baseline version of query corrector relies on *noisy channel ranker*, which select candidate for correction according to

$$\hat{c} = \arg\max_{c \in C} P(q|c)P(c) \tag{6.10}$$

where the C is the set of relevant canditates to query q.

The main issue of *noisy channel* approach is that it relies on the probability distributions (which are estimated from noisy source) too heavily and doesn't allow to incorporate other potentially useful signals, such as presence in (refined, clean) dictionary or phonetic similarity of query/candidate pair.

Useful tool is a re-ranker. We take e.g. top 10 candidates according to noisy channel ranker and create a new permutation of candidates using ranking function h and we select the top-ranked candidate from the new ordering. The task can be formalized as follows.

For given query q find correction $\hat{c}$ such that

$$\hat{c} = \arg\max_{c \in C} h(\phi(q, c)) \tag{6.11}$$

One can see that equation (6.10) is the special case of (6.11).

6.5 Learning to rank approaches

Current state of learning to rank task may be divided into three different approaches. These approaches differ in what kind of loss function they optimize [15]. It can be divided into three major categories. The first kind of a loss functions is a *pointwise* loss. This approach ignores a group structure of candidates for given query and tries classify whenever the vector is relevant or not. Another approach is a *pairwise* loss. We construct pairs of original feature vectors a then we tried to minimize number of wrong ordering within the pairs. Last kind of losses is referred as a *listwise* loss. This loss function directly minimizes an upper bound of the evaluation measure error mentioned above.

6.5.1 Pointwise approach

In this approach the ranking task is transformed to a binary classification. The task is to classify given feature vector as relevant or non-relevant, thus conventional classification tools can be used.

We have used a *logistic regression* classifier with regularization. The learning task is to solve following unconstrained optimization problem

$$\min_{\mathbf{w}} -\sum_{i=1}^{m} y_i \log h_{\mathbf{w}}(\mathbf{x}_i) + (1 - y_i)\log(1 - h_{\mathbf{w}}(\mathbf{x}_i)) + \frac{\lambda}{2}\|\mathbf{w}\|^2 \tag{6.12}$$

where m is number of all candidates for all the queries in training set, λ is regularization factor - the trade off between hypothesis complexity and classification performance on the train set and y_i is label of corresponding feature vector $\mathbf{x}_i$. The $h_{\mathbf{w}}(\mathbf{x})$ is logistic regression function defined as

$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}} \tag{6.13}$$

The main problem of the pointwise approach is, that it doesn't preserve a group structure of the candidates for specific query and treats all candidates as the same. One can see, that bias to certain query candidates is potential threat, as seen in Figure 7.

6.5.2 Pairwise approach

Pairwise approach transforms ranking as a binary classification problem. It classifies pairs of feature vectors $\mathbf{x}_i$, $\mathbf{x}_j$ into $\mathcal{Y} = \{+1, -1\}$. $y_i = +1$ when vector $\mathbf{x}_i$ should be ranked ahead of $\mathbf{x}_j$ and $y_i = -1$ vice versa.

Consider following traning set $T = \{(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}),\ y_i\}_{i=1}^{m}$ where each instance consists of a pair of feature vectors created from all possible pair permutations within associated candidate set $\mathcal{C}_i$ for given query q_i. Let k be number of queries in the training set and suppose that we wish to re-rank (at most) top n candidates for every query. When we omit pairs of the same vectors, it gives us total amount $m \leq k(n^2 - n)$ of training pairs.

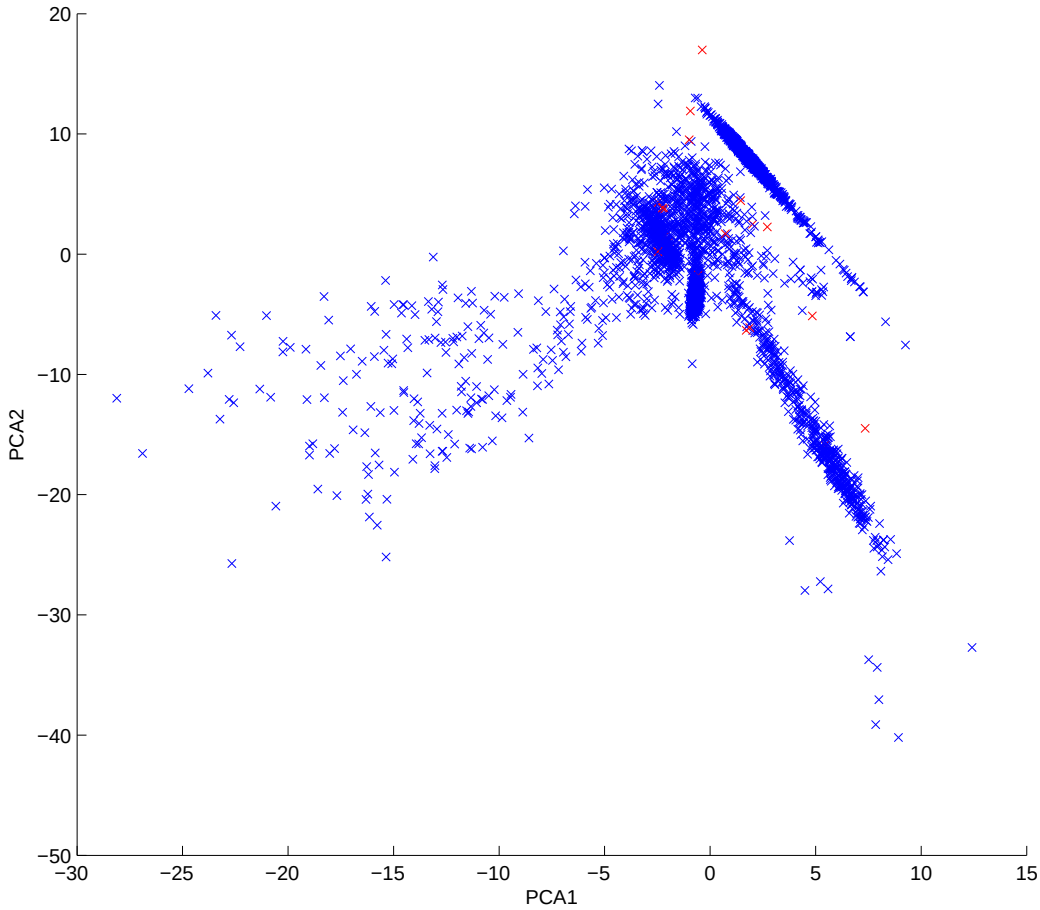


Figure 7: Pointwise approach visualized using major PCA components of bigram query-candidates features space (relevant candidates in red)

In order to learn to rank we have to solve following unconstrained optimization problem [15]

$$\min_{\mathbf{w}} \sum_{i=1}^{m} \max\{0, 1 - y_i \langle \mathbf{w}, \mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)} \rangle\} + \frac{1}{2C} \|\mathbf{w}\|^2 \tag{6.14}$$

which minimalizes regularized *hinge loss* function. C is a parameter trade off between hypothesis complexity and performance on training data set. This algorithm is referred as *Ranking SVM*. Essentially, it is a soft-margin SVM classifier on the set of pairs of feature vectors. The goal is to minimalize average number of inversion of vectors $\mathbf{x}_i^{(1)}$ and $\mathbf{x}_i^{(2)}$ in ordering over the training set.

By using the theorem [4] that a non-negative combination of a convex functions (both vector norm and maximum of affine functions are convex functions) is a convex function we

can conclude that problem (6.14) is convex. Since the objective function is convex we are guaranteed that every local minimum is also a global minimum [4]. Strictly speaking, since it's not differentiable, we cannot [4] use the gradient descent method nor the Newton's method, although *subgradient methods* can be used in similar way.

Let us derive subgradient of the objective function of (6.14)

$$\nabla L(\mathbf{w}) = \sum_{i=1}^{m} \frac{\partial l_i}{\partial \mathbf{w}} + \frac{1}{C}\mathbf{w} \tag{6.15}$$

$$\frac{\partial l_i}{\partial \mathbf{w}} = \begin{cases} -y_i\langle \mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)}\rangle & \text{if } y_i\langle \mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)}\rangle \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{6.16}$$

Then, we can apply a gradient descent algorithm described as Algorithm 3. Since the size m of the training set can be very big, it's convinient use a *stochastic gradient descent* in practice.

Algorithm 3: Optimization of hinge loss using gradient descent

input : training set $T = \{(\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)}), y_i\}_{i=1}^{m}$
output: optimal $\mathbf{w}$

```
w_0 ← 0
Δ ← +∞
while Δ > ε do
    w_{k+1} ← w_k − α_k ∇L(w_k)
    Δ ← L(w_k) − L(w_{k+1})
end
```

Candidates are ranked according to their scores by function $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. This function represents a total order on the set of feature vectors (precisely on the equilance classes). The geometrical interpretation behind (6.14) is visualised in Figure 8 and in Figure 9. We are looking for the hyperplane $\mathbf{w}^T\mathbf{x} = 0$, which separates positive instances (red) and negative one (blue) in order to achieve maximal margin between samples of opposite instances. Candidates are ranked according to length of their projection onto vector $\mathbf{w}$.

6.5.3 IR SVM

Previously mentioned approach has some disadvantages. The problem is, that hinge loss function used in (6.14) assigns the same loss value for the inversion in ranking on the top of the list as for the bottom ranked candidates [5]. In our task, where arguable the most relevant measure of quality of the ordering is $P@1$, it is undesired treating losses the same.

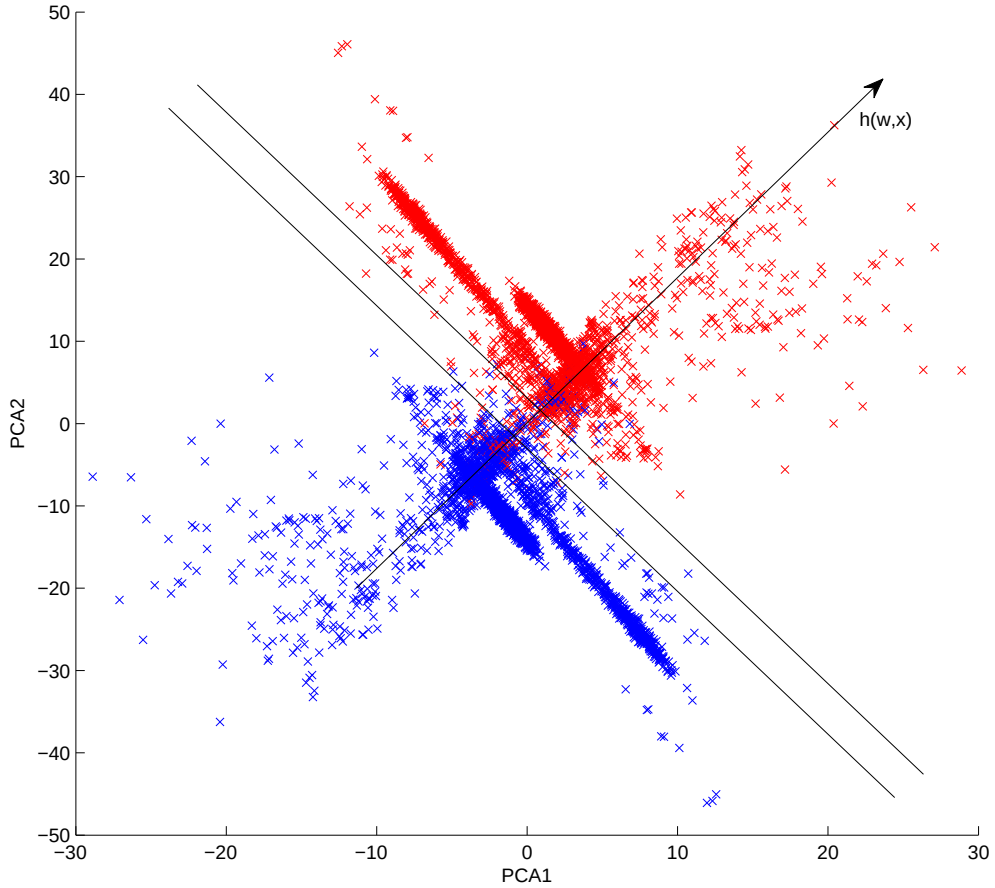


Figure 8: Pairwise approach visualized using major PCA components of bigram query-candidates features space

Moreover, greater number of candidates for single query can bias resulting model towards to this case [15].

[5] proposed improved model of pairwise learning to rank algorithm by modifying used hinge loss function as

$$\min_{\mathbf{w}} \sum_{i=1}^{m} \tau_{k(i)} \mu_{q_i} \max\{0, 1 - y_i \langle \mathbf{w}, \mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)} \rangle\} + \frac{1}{2C} \|\mathbf{w}\|^2 \tag{6.17}$$

where $\mu_{q_i} = \frac{1}{|\mathcal{C}_i|}$ and $\tau_{k(i)}$ represent weight of the i-th instance of k-th rank type. One of the method of finding value of $\tau_{k(i)}$ is based on the simulation as described in [5]. Firstly, we find for all the queries their perfect ranking permutation (according to labeled data, so e.g. $P@1 = 1.0$). Then we swap positions of randomly chosen candidates associated

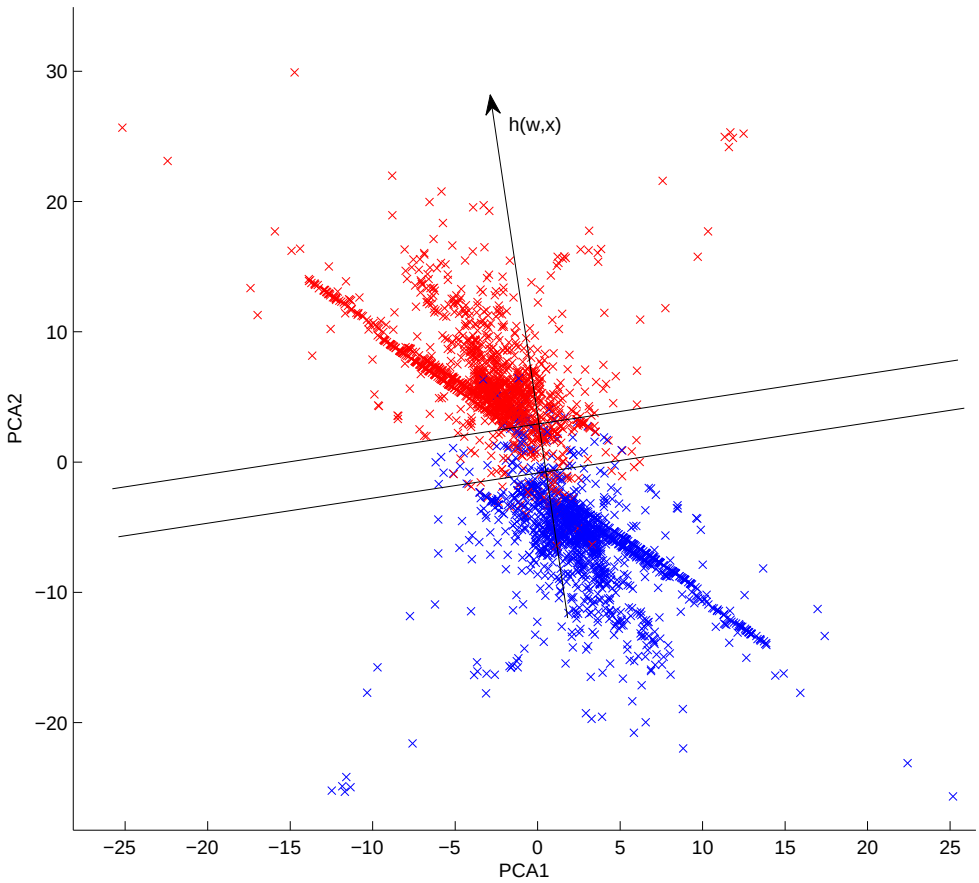


Figure 9: Pairwise approach visualized using major PCA components of unigram query-candidates features space

with ranking of k-th type and calculate evaluation metrics for the new permutation. There is usually drop in terms of evaluation metrics. Then, the drops across all the queries are averaged and it is value of parameter for the rank pair.

The result is, that we calculate loss for each instance individually which arguably help us to achieve better ordering since pairs with greater impact of evaluation metrics have greater loss then other one. Visualisation of modified hinge loss function can be seen in Figure 10.

6.5.4 Listwise approach

Oppose the previous algorithms, the listwise approach directly optimize an evaluation measure (MAP, NDCG) on the test sample. Usually such measures aren't convex or even

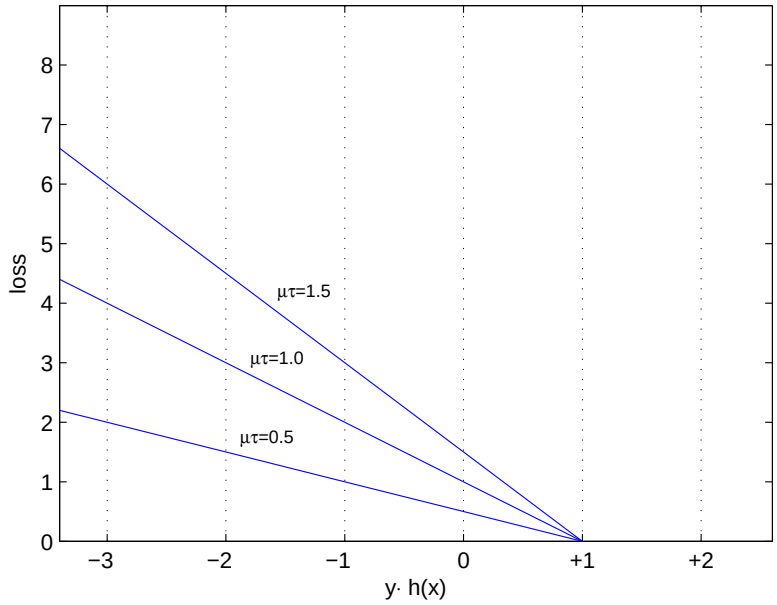


Figure 10: Hinge loss with different slopes

continuous functions so the problem of finding optima is hard. Although an upper bound of such loss function can be found (even a convex one) [15].

Current state of the art ranking algorithms are from this category, such as Lambda Trees [12], SVM MAP, AdaRank [27] and others. These algorithms are able to significantly outperform [27] the ones based on the pairwise principle.

These algorithms weren't applied to presented task and so they aren't part of the thesis.

6.6 Feature engineering

Feature engineering is the task of designing features which are employed into ranking problem (or generally any machine learning task). In this task, the knowledge of domain of the model and data are very helpful. Ideally, the goal is to design such features, whose values aren't correlated and shares similar scale – it's known, that feature scaling greatly affects performance of training SVM classifier (searching for support vectors) or using gradient method (speed of convergence).

In runtime, we need to compute values of designed features first. It usually requires external data source which adds additional information to ranking process. Then the feature vectors are supplied into a ranking model which is trained as mentioned above. The output is a permutation of objects. Diagram in Figure 11 shows you such ranking model.

We are using features drawn from these categories

a) probabilistic features

b) meta-features

c) lexicographical features

d) surface-form features

e) phonetic similarity features

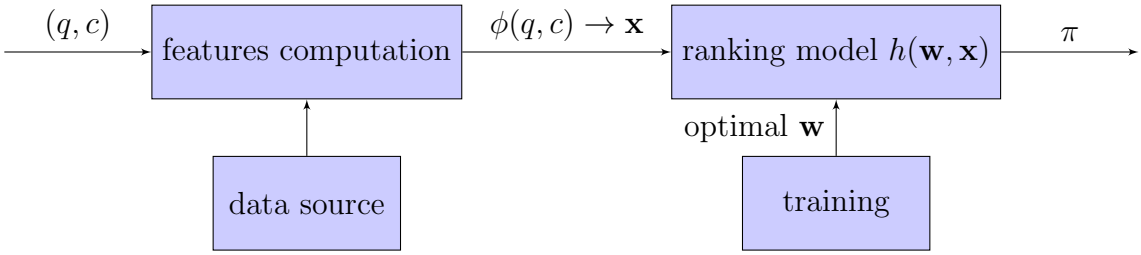


Figure 11: Ranking model

6.6.1 Probabilistic features

These features are derived from probabilistic models. We are using n-gram LM probability of candidate $\log P(c)$ and original query $\log P(q)$ and the error model score $\log P(q|c)$.

6.6.2 Meta-features

Meta-features are maps of existing features providing additional dimensionality of the problem of the ranking which allows creating non-linear hypothesis in the original space. It is known [12] that meta-features helps build to less complex hypothesis with better performance.

We are using *noisy ranker* score, calculated as $P(c)P(q|c)$ and feature calculated as $P(q|c)P(c)/P(q)$.

6.6.3 Surface-form features

These features indicates, whether original query q and candidate c differ in certain string patterns, such as q is start of c (users hit enter key prematurely). Some of them are:

Starts with, ends with This feature fires, whenever query q is starting character sequence of candidate c. The idea behind it is that we want to capture the event, when a user hit enter key accidentally before finish typing query. A classical example is query `goog`, where is desired to bring candidate `google` (which is in edit distance 2) to the top.

Common misspells patterns From query reformulation session we have observed the pattern, that many very popular search terms are being misspelled and then corrected. Since the error model is estimated from the same source of data, one could think that these misspells will be already captured in error model and won't penalizete these errors too heavily. Is may not be always true. For example the term `simsnovi` should be corrected to `simpsonovi` but it requires costly operations.

We have compile list of the most common misspelling and their corrections that are in maximal edit distance 2 apart. Presence of query/candidate pair in that list is captured by this feature.

List of top 10 common misspelling is listed in Table 5.

$q \rightarrow c$	occurrence
`facebok` → `facebook`	1033771
`yotube` → `youtube`	566195
`cz` → `com`	408184
`yutube` → `youtube`	299593
`faceboo` → `facebook`	289987
`com` → `cz`	280641
`fecebook` → `facebook`	261568
`gogle` → `google`	248969
`ww` → `www`	227772
`youtobe` → `youtube`	202224

Table 5: Top 10 common misspelling

Numerical terms When q and c differ in numerical terms, this feature fires. For example, when a user searches for specific product (like `iphone 2`), it is undesiring to suggest different (although much more popular) like `iphone 5` product from the same lineup.

Missing TLD When q starts with `www` and does not contains any TLD (top level domain) and candidate c does, the feature fires. We assume that when the query starts with `www` it is a URL which can't be valid since doesn't contain any TLD. It helps penalizate queries like `www.wikipedia` and so on.

6.6.4 Phonetic similarity

Phonetic similarity of two strings is a equivalence relation, which breaks a set of finite length strings from finite alphabet into equivalence classes. All strings within class are phonetically similar to each other.

Usual approach is to create s mapping which maps a string into its *phonetic code*. When two strings shares the same phonetic code, they are in phonetic similarity relation.

We use *Double Metaphone*[5] algorithm which represent previously mentioned mapping. It actually produces two codes - primary and secondary one. This version of Metaphone algorithm works well even for many non-English words.

Example: Word similarity

$$\begin{aligned} phon(\texttt{kobila}) \to \texttt{KPL},\ phon(\texttt{kobyla}) \to \texttt{KPL} &\implies \texttt{kobila} \sim_P \texttt{kobyla} \\ phon(\texttt{gugl}) \to \texttt{KKL},\ phon(\texttt{google}) \to \texttt{KKL} &\implies \texttt{gugl} \sim_P \texttt{google} \end{aligned}$$

This feature fires when query q and candidate c are in the same equivalence class.

6.6.5 Lexical features

Using map-reduce algorithm we have compiled a histogram of words used in Czech Wikipedia. The task has been computed on in-house built cluster with map-reduce framework implementation `mincemeat-py`[6].

We used features like presence in Wikipedia article title or cumulative logarithmic occurrence of term in article bodies averaged by number of terms in query.

6.7 Addtional work

Area of learning to rank is very active. Many other methods were proposed. E.g. [6] proposed usage of multiple ranking models. First (a listwise one) will focus on hard cases and the other one (sigmoid loss function) will be trained to achieve high accuracy on top of the ordering.

Unfortunately, the sigmoid loss function isn't convex, so it has (typically) multiple local optima and it is hard to optimize it.

[5] http://www.atomodo.com/code/double-metaphone/metaphone.py/
[6] https://github.com/michaelfairley/mincemeatpy

7 Runtime pipeline

Proposed system has been implemented in Python programming language version 2.7. Training of a ranking models was done by Mathworks MATLAB software. Whole pipeline can be viewed in Figure 12.

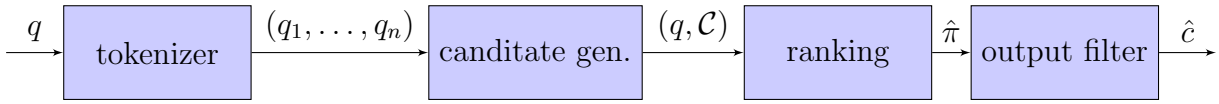


Figure 12: Runtime pipeline

7.1 Tokenization

The input of this component is a query string $q \in \Sigma^*$. The task is to output n-tuple of terms (words) inside of the q. Formally speaking, we are looking for a map

$$l : \Sigma^* \to \Sigma^{*n} \tag{7.1}$$

Such process is necessary for suggesting better candidates – we are suggesting candidates for terms in query, not a whole query (since itself can be very unique). We used three types of tokenizer - one based on simple tokenization of words separated by spaces, the second one by dots (used in URLs) and the third is probabilistic tokenizer which deals with wrong word segmentation - when user didn't separate words with spaces correctly.

We have adopted algorithm described in [19] with slight modification on penalizating smoothed OOVs tokens. [13] showed that type used smoothing has noticeable impact on tokenization precision. We have not adopt this approach and used default smoothing (the same LM, *11m@4gram+queries*).

Example: probabilistic segmentation Suppose we apply tokenization on user's input $q =$ `wwwauto`. Algorithm 4 generates a set of all possible partitioning with single space insertion

```
w wwauto
ww wauto
www auto
...
wwwaut o
```

From this set we pick the most probable segmentation under n-gram model, e.g.

$$P(\texttt{www}, \texttt{auto}) = P(\texttt{www}|\texttt{<s>})P(\texttt{auto}|\texttt{www})P(\texttt{</s>}|\texttt{auto})$$

we grab the first half and apply the same procedure on the rest of string q. Since the unigram `auto` is popular, the algorithm ends.

Algorithm 4: Probabilistic segmentation

input : string q
output: n-tuple of terms $Q = (q_1, q_2, \ldots, q_n)$

```
while q ≠ Λ do
    l ← |q|
    S ← ∅
    for j ≤ l do
        q_1 ← q_1^j
        q_2 ← q_j^l
        S ← S ∪ {(q_1, q_2)}
    end
    (q_i, q_{i+1}) ← arg max_{(q_i,q_{i+1})∈S} P(q_{i+1}|q_i)
    Q ← Q + q_i
    q ← q_{i+1}
end
```

7.2 Candidate retrieval

The input of this component is ordered n-tuple of terms from the tokenizer. The task is to create a set of candidates $\mathcal{C}$, from which correction candidates are drawn to a ranking process. Such set is constructed as

$$\mathcal{C} = cand(q_1) \times cand(q_2) \times \ldots \times cand(q_n) \tag{7.2}$$

where

$$cand(q) = \{c \in \tilde{D} \,|\, sim(q, c) \geq \tau\} \tag{7.3}$$

where q is term searched by users, $c \in \tilde{D}$ is a candidate from a noisy dictionary and $sim(q, c)$ is a string similarity measure of q and c.

In our implementation we use following formula for computing candidate set

$$cand(q) = \{c \in \tilde{D} \,|\, jaccard(q, c) \geq \tau(q)\} \tag{7.4}$$

where $jaccard(q, c)$ is *Jaccard* similarity measure of strings q and c. The $\tau(q)$ is the minimal similarity required for a string c to be in a set. This bound is a function of length of q, typically set about 0.5.

Example: Jaccard similiary Suppose we have words $w_1 =$ `elegant`, $w_2 =$ `elephant`. Under letter bigram representation we have

$$\begin{aligned} w_1 &= \{\texttt{\$e}, \texttt{el}, \texttt{le}, \texttt{eg}, \texttt{ga}, \texttt{an}, \texttt{nt}, \texttt{t\$}\} \\ w_2 &= \{\texttt{\$e}, \texttt{el}, \texttt{le}, \texttt{ep}, \texttt{ph}, \texttt{ha}, \texttt{an}, \texttt{nt}, \texttt{t\$}\} \end{aligned}$$

Then Jaccard similiarity of w_1 and w_2 is given by

$$jaccard(w_1, w_2) = \frac{|w_1 \cap w_2|}{|w_1 \cup w_2|} = \frac{6}{11} = 0.5455 \tag{7.5}$$

For fast fuzzy string retrieval, we use tool *Simstring* [20]. Data source for the dictionary is large number of the most popular unigram compiled from the web pages containing $|\tilde{D}| \approx 15$ M entries.

The cardinality of a set $\mathcal{C}$ resulting by using formula (7.2) could be very large, especially when n is large ($n \geq 3$), since it is given by

$$|\mathcal{C}| = \prod_{i}^{n} |cand(q_i)| \tag{7.6}$$

Processing such large number of candidates would take tens to hundreds of seconds which is unacceptable.

In Figure 13 we see a histogram of logprobs of all candidates for certain sample query. The Figure 14 shows you a histogram of logprobs of candidates, which have probability equal or greater then originally typed term. So one can see, that combining all possible candidates for all terms in query results in large number of non-sense combinations which arguably can't be valid candidates.

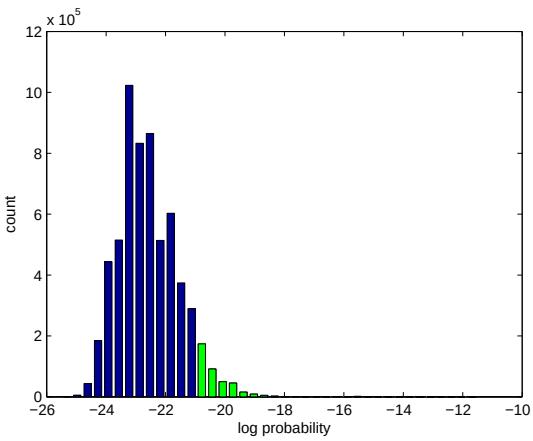


Figure 13: Full candidate histogram

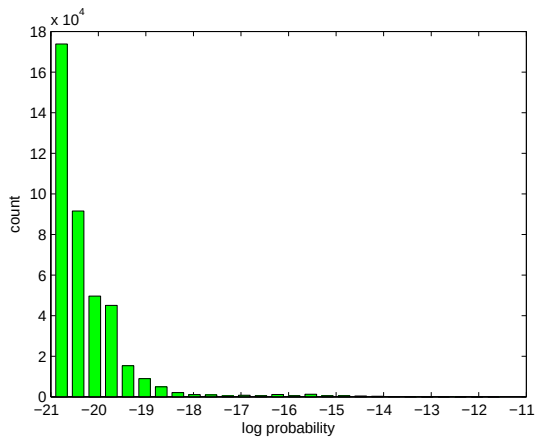


Figure 14: Histogram of pruned candidates

To deal with it, we have proposed a pruning algorithm which significantly reduces number of candidates during construction a set $\mathcal{C}$. One can view process of construction $\mathcal{C}$ as a building a tree with depth of $n + 1$ where each node contains candidates of q_{n-1} term. Candidate for whole query is represented by path from root to leaves in this tree.

Since probability of the sequence $c_1, c_2, \ldots, c_n$ under n-gram model is not increasing function in terms of n (e.g. inequality $P(c_1, c_2) \geq P(c_1, c_2, c_3)$ always holds) we can terminate computation without reaching leaf of the tree (visiting all the candidates).

Essentially, it is a depth-first strategy with heuristic on probability in n-gram model. The code is listed as Algorithm 5.

Algorithm 5: Candidates prunning

```
input  : n-tuple of terms Q = (q_1, q_2, ..., q_n)
output: pruned candidates set C

function prune(c, Q, i)
if |Q| = 0 then
    return c
end
else
    forall the q ∈ C_i do
        p ← P(c, q)
        if p < h then
            continue
        end
        else
            l ← l ∪ prune((c, q_i), (q_{i+1}, ..., q_n), i + 1)
        end
    end
    return l
end
h ← P(q_1, q_2, ..., q_n)
for i ≤ n do
    C_i ← cand(q_i)
end
C ←prune(Λ, Q, 1)
```

In a runtime this pruning strategy yield into $100\times - 1000\times$ computation speedup without omitting a candidates with greater probability from a ranking process.

Other performance optimization is that $|\mathcal{C}|$ has an upper bound. We set empirically this value to $1 \cdot 10^6$. When $|\mathcal{C}| > 1 \cdot 10^6$ (after pruning process), we incrementally rise similarity bound τ for candidates q_i, when $q_i = \arg\max |cand(q_i)|$. This process is iteratively treat until $|\mathcal{C}| \leq 1 \cdot 10^6$.

Real life results shows, that sequence of $|\mathcal{C}|$ quickly converges to desired criterion, thus only a small number of iteration is typically made.

In addition, all strings within edit distance 1 from q are added to candidate set, as mentioned in chapter 2.

In fact, the things are even more complicated – we use different candidate retrieval component for queries which are URLs - for them we suggest for example different TLDs, domain names and (if present) corrupted `www` start.

7.3 Ranking

Ranking process follows mentioned IR SVM approach without τ coefficient where query / candidate feature pair contains 15 elements. A linear hypothesis was trained on a test set from each dataset separately for 1-grams, 2-brams, 3-grams and 4-grams. Each of these models is parametrized by a C, which is trade off between in-sample classification error and soft-margin width. Final model parameter C was selected by a performance on the validation set.

These resulting four hypothesis functions are applied separately on the test data according to query n-gram order.

7.3.1 Language model

We used a language model *11m@4-gram+queries* which achieves 701 perplexity on a test queries (see chapter 4). In runtime, it takes about 13 GB in memory on a 64-bit architecture.

7.3.2 Error model

We used the Brill's & Moore's error model with a size of window $L = 2$. Model consisted of approximately 158 K transfemes.

Moreover, in case of query is an URL, some adjustment were made. For example we lowered penalization of insertion and deletion of characters like `.` or `-`.

7.3.3 Other features

The total amount of features is 15. The list of common misspelling contains 46 K entries. Wikipedia title list contains about 400 K entries and cutted histogram of words in article bodies has 540 K entries.

7.4 Output filter

This component checks resulting correction suggestion if it differs in certain patterns from original query (such as lower vs. upper case, accents presence, etc.) and in these cases bans candidate and returns original query.

8 Evaluation and results

Proposed system was evaluated on the test data provided by Seznam.cz. We posses two different datasets (one referred as the *old* and other one referred as *new*). On these datasets we have used scheme train-valid-test and showed, that we statistically significantly outperformed a baseline noisy channel model.

Proposed system was tuned into high precision rather than high recall, since we assume that suggesting a wrong correction frequently may be more frustrating for users rather than not giving a suggestion for correction in certain situations.

For obtained results we performed a *McNemar's test* to ensure a statistical significance.

8.1 Metrics

We evaluate criteria defined as follows

$$\begin{aligned}
\text{accuracy} &\stackrel{\text{def}}{=} \frac{\#\text{ relevant outputs}}{\#\text{ queries}} \\
\text{precision} &\stackrel{\text{def}}{=} \frac{\#\text{ relevant corrections}}{\#\text{ proposed corrections}} \\
\text{recall} &\stackrel{\text{def}}{=} \frac{\#\text{ relevant corrections}}{\#\text{ required corrections}}
\end{aligned}$$

The definitions are consistent with [14].

8.2 The old dataset

This dataset was created by human annotators and contains 5 K pairs of (query, desired correction). Usage was following: 60% is the train data, 20% validation data and 20% test data. These sets are non-overlapping.

From 5 K of queries is about 7% marked as incorrect (desired correction is not equal to original term), although we have noticed, that sometimes we can see evident misspell which is not corrected (leaved as it is). So we view this data as a noisy one.

8.2.1 Model selection

From the test data we have trained a few models which vary in parameter C. From these models we selected the one that minimizes error on the validation dataset. Then, resulting model was retrained on train+valid data and evaluated on the test data.

	noisy channel	proposed system
accuracy	0.897	**0.943**†
precision	0.245	**0.761**†
recall	0.238	**0.285**†

Table 6: Results on the old dataset. † denotes statistically significant results (p-value $<$ 0.05)

8.2.2 Results

Obtained results cannot be directly compared to other works, since (in 2013) in our knowledge we aren't aware of similar research performed on the Czech query data. The other fact is, that we have to correct both Czech and English spelling (in some cases even german words), so the task is even more complicated than similar research across the world. The other thing is, that when we manually checked reported errors made by our system we observed, that in fact many of them aren't errors in our opinion, but just doesn't match annotated correction. In many cases, we were asked to give correction with accents mark and in different cases we weren't supposed to add accents (e.g. `kocka` → `kočka` is not in some cases a valid correction). Since previous mentioned, we banned correction consist of accents marks and in the other case in addition, we check strings equality even withount accents marks. All kinds of reviously mentioned noises in the data contributes to lower performance, so we assume, that in fact our proposed system achieves even more performance than evaluated.

But some numbers can picture an idea - for example, in 2006, a group of researches from Microsoft and Norteastern and Tianjin University reported [16] 0.89 accuracy. In 2010, researches from Microsoft reported, that their system achieved 0.916 accuracy [14]. Research team consist of 9 people from University of California in 2011 reported [13] that they achieved 0.9482 accuracy. Unfortunately, we cannot compare with Google results as published [26], since they used custom evaluation metrics and criterion and have not provided enough data to calculate their accuracy.

Implementation manages to process 1 query per 1.82 s on average on the single processor.

8.3 The new dataset

This dataset was created after the first one. It consist of 100 K queries, where for each query is one or more possible corrections given by a human annotator. Once again, for training we divided dataset into three non-overlapping sets: 60% train set, 20% validation set and 20% test set. In resulting test set is about 6.5% of queries marked as incorrect. But it contains significant number of queries, which are obviously misspelled, but labeled as when no correction is required as well as inconsistently labeled queries - the same or

similar types of queries are labeled differently. This mentioned above contributes to a lower than real accuracy

For this dataset, we have asked researches from Seznam.cz to test our system independently. They used their private test data, which we have never seen and were asked to give a performance evaluation. In this process, we have been involved only in setuping publicly available system, which they queried over the network. After this testing procedure, we obtained the results.

8.3.1 Model selection

Once again, models were trained on the test data for a set of parameters C varying from 0.05 to 3. Our experience and analysis determinated that maximum of model's performance lies within this range.

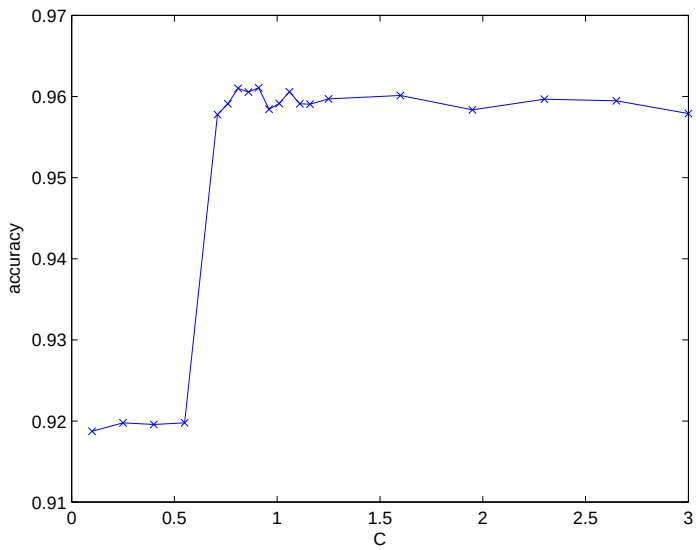


Figure 15: Validation error of bigram ranking model varying in C

8.3.2 Results

We performed two evaluations – one on the test subset of the new dataset (results in Table 7) and the other one performed by researches from Seznam.cz on their test set (results in Table 8).

One can see, that results reported by Seznam.cz's researches are slightly different from the ones performed on our test set. It can be explained by various reasons. The first one is, that their test set is not drawn from the same distribution as our dataset. In that case, we cannot say anything about the performance of the proposed system based on our

	noisy channel	proposed system
accuracy	0.9157	**0.9473**†

Table 7: Results on the new dataset. † denotes significant results (p-value < 0.05)

	Seznam's research prototype	proposed system
accuracy	**0.9494**	0.9404

Table 8: Seznam test evaluation

out-of-sample performance estimation (results on our test set), thus resulting performance may vary. The second reason may be, that we have used slightly different testing protocol. As we already mentioned, in some cases, the queries are arguably poorly annotated, thus we made following two adjustments – the annotated (considered as *ground truth*) candidates containing word `sexi` we turned into (arguably correct representation) `sexy` and for the queries in form of URL, where desired candidates were marked as e.g. `www.ceskatelevize.cz` we replaced it with `www.ceskatelevize.cz` (when the candidate is URL witnout any additional words, we join it into a single term so it could be a valid URL).

The reasons for this adjusment of the test set are obvious – we didn't want to fit the noise in the data, but rather fit the target hypothesis well and in addition, some of our features were designed to deal with these kind of things, so they would be meaningless. One could came up with additional corrections of the test set, but it would be time counsuming and difficult, so we applied only these.

9 Conclusion

The task of the query spelling correction turns out to be very difficult. Actually, it can be divided into two subtask - spellchecking and autocorrection. The spellchecking can be viewed as a binary classifier which determines, when the typed query has to be corrected by an autocorrection or not. It has to be extraordinary precise, since (in our examined Czech data) only about 7% of queries are marked as misspells (in English it is often about 15%). Then, when the query is marked as misspell, an autocorrection model have to suggest different string which is drawn from a very large space. Moreover, when the correct query is misclassified (marked as misspell) it's guaranteed that accuracy performance drops. Both mentioned components have to perform very well in order to achieve a good performance of the system.

Our proposed system was optimized into high precision – that means, when it suggest a correction, it is often a correct one. The price for that is, that often the system isn't confident enough to pull of suggested candidate and rather leave original query unchanged. This behaviour is measured by recall.

Proposed system does arguably well in precision but has lower performance in recall. It can be explained by two things - firstly as we already said, we maximized precision which leads to lower recall. Secondly, the task of creating a set of candidates turns out not to be trivial. We examined, that significant portion of not corrected misspells is caused by a lack of presence desired correction in candidate set. Our implemented solution of construction of the candidate set can be considered as the weakest link in the chain.

Our system can be easily tuned into better performance by suppling more data – especially query histograms, from which the language model is build. As we shown in chapter 4.4.3, our language model have higher perplexity than similar model described in [14] for English. The reason may be, that our language model suffer from insufficient size of the data or the Czech queries (and language) are just more complex than English.

The other possibility is to implement a better ranker. It is reported [1] that simple logistic regression classifier built on top of the powerful ranker achieves significant improvement in question answering, which is kind of similar task.

The next way in improvement could be to incorporate more features. In proposed model we have used only 15 features, although e.g. [13] reported that they used 89 raw features and additional 49 metafeatures. We didn't follow this way since we didn't have enough train data to fully supplement such large number of model's parameters and we wish to avoid overfitting. In addition, it would be helpful to implemenent Brill's & Moore's error model with $L = 3$ and with positional information.

We have showed that our proposed model statistically significantly outperformed a baseline model and performed comparably well in terms of accuracy with current Seznam.cz's research prototype. We hypothesise that combination of those two models would result in significant performance gain and would be a beneficial for users.

References

[1] Arvind Agarwal, Hema Raghavan, Karthik Subbian, Prem Melville, Richard D Lawrence, David C Gondek, and James Fan. Learning to rank for robust question answering. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 833–842. ACM, 2012.

[2] Farooq Ahmad and Grzegorz Kondrak. Learning a spelling error model from search query logs. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 955–962. Association for Computational Linguistics, 2005.

[3] Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71, 1996.

[4] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[5] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. Adapting ranking svm to document retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 186–193. ACM, 2006.

[6] Vitor R Carvalho, Jonathan L Elsas, William W Cohen, and Jaime G Carbonell. A meta-learning approach for robust rank learning. In *SIGIR 2008 workshop on learning to rank for information retrieval*, volume 1, 2008.

[7] Qing Chen, Mu Li, and Ming Zhou. Improving query spelling correction using web search results. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 181–189, 2007.

[8] Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics, 1996.

[9] Fred J Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[10] Robert C. Moore Eric Brill. An improved error model for noisy channel spelling correction. *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics, ACL' 00*, 2000.

[11] Stefan Evert and Marco Baroni. Testing the extrapolation quality of word frequency models. In *Proceedings of Corpus Linguistics*, volume 2006, 2005.

[12] Yasser Ganjisaffar. *Tree Ensembles for Learning to Rank*. PhD thesis, University of California, 2011.

[13] Yasser Ganjisaffar, Andrea Zilio, Sara Javanmardi, Inci Cetindil, Manik Sikka, Sandeep Katumalla, Narges Khatib, Chen Li, and Cristina Lopes. qspell: Spelling correction of web search queries using ranking models and iterative correction. In *Spelling Alteration for Web Search Workshop*, page 15, 2011.

[14] Jianfeng Gao, Xiaolong Li, Daniel Micol, Chris Quirk, and Xu Sun. A large scale ranker-based system for search query spelling correction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 358–366. Association for Computational Linguistics, 2010.

[15] LI Hang. A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems*, 94(10):1854–1862, 2011.

[16] Mu Li, Yang Zhang, Muhua Zhu, and Ming Zhou. Exploring distributional similarity based models for query spelling correction. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 1025–1032. Association for Computational Linguistics, 2006.

[17] Nitin Madnani. Querying and serving n-gram language models with python. *The Python Papers*, 4(2):2009, 2009.

[18] Roger Mitton. Fifty years of spellchecking. *Writing Systems Research*, 2(1):1–7, 2010.

[19] Peter Norvig. Natural language corpus data. *Beautiful Data*, pages 219–242, 2009.

[20] Naoaki Okazaki and Jun'ichi Tsujii. Simple and efficient algorithm for approximate dictionary matching. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 851–859. Association for Computational Linguistics, 2010.

[21] Ronald Rosenfeld. Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278, 2000.

[22] Claude E Shannon. Prediction and entropy of printed english. *Bell System Technical Journal*, 30(1):50–64, 1951.

[23] Andreas Stolcke, Jing Zheng, Wen Wang, and Victor Abrash. Srilm at sixteen: Update and outlook. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop*, 2011.

[24] Kristina Toutanova and Robert C Moore. Pronunciation modeling for improved spelling correction. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 144–151. Association for Computational Linguistics, 2002.

[25] Tomáš Werner. *Optimalizace (in czech)*. CTU FEE, 2013.

[26] Casey Whitelaw, Ben Hutchinson, Grace Y Chung, and Gerard Ellis. Using the web for language independent spellchecking and autocorrection. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 890–899. Association for Computational Linguistics, 2009.

[27] Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391–398. ACM, 2007.

[28] George Kingsley Zipf. Human behavior and the principle of least effort. 1949.

Appendix A

Used mathematical symbols and notation

Used notation follows standard conventions. Definitions of the optimization task and related problems mainly follows [25].

Vector

A real vector $\mathbf{x} \in \mathbb{R}^n$ is a matix $n \times 1$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{A.1}$$

where entries $x_1, \ldots, x_n \in \mathbb{R}$.

The inner product

Consider vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. Their *inner product* is denoted by

$$\mathbf{x}^T\mathbf{y} = \langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{n} x_i y_i \tag{A.2}$$

Both notations are used interchangeably when its convenient.

The alphabet, strings and characters

Alphabet Let $\Sigma = \{a, b, c, \ldots, z, \_, \Lambda\}$ be the alphabet. Symbol Λ denotes the *empty* character. Σ^* denotes a set of all finite-length strings on the alphabet Σ. Element $\alpha \in \Sigma$ is called *character*.

Concatenation A *Concatenation* of strings $c \in \Sigma^*$ and $p \in \Sigma^*$ is denoted by $cp \in \Sigma^*$

Strings A *string* is the concatenation of finite number of characters. Let $s \in \Sigma^*$ be a string. Then $s_i \in \Sigma$ denotes the i-th character of the string s.

The maximum, minimum and argument

Let $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is arbitrary set. Let

$$Y = f(\mathcal{X}) = \{f(x) \mid x \in \mathcal{X}\} \tag{A.3}$$

be the image of set $\mathcal{X}$. The *minimum of function f on $\mathcal{X}$* is

$$\min_{x \in \mathcal{X}} f(x) = \min Y \tag{A.4}$$

when minimum of Y exist. The *argument* of minimum (A.4) is

$$\hat{x} \in \arg\min_{x \in \mathcal{X}} f(x) = \{\tilde{x} \in \mathcal{X} \mid f(\tilde{x}) = \min_{x \in \mathcal{X}} f(x)\} \tag{A.5}$$

Technically $Y_f = \arg\min_{x \in \mathcal{X}} f(x)$ is a set. Sometimes we abbuse notation by saying that $\hat{x} = \arg\min_{x \in \mathcal{X}} f(x)$, meaning $\hat{x}$ is an arbitrary element of Y_f.

Definition for max is analogous.

Logarithm

When not mentioned otherwise, then we use decadic logarithm by default.

$$\log x = \log_{10} x \tag{A.6}$$

Vector norm

Let $\|\cdot\|$ be the vector norm defined as

$$\forall \mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| = (x_1^2 + \ldots + x_n^2)^{1/2} \tag{A.7}$$

so we are using l_2 vector norm when not mentioned otherwise.

Appendix B

CD Content

In Table 9 are listed names of all root directories on CD

Directory name	Description
`bp`	thesis in pdf format.
`sources/py`	source codes
`sources/matlab`	m-files (optimization, etc.)
`sources/tex`	thesis in Latex source code
`models`	trained models for Learning to Rank

Table 9: CD content