Text Analysis and Retrieval 2014

Course Project Reports

University of Zagreb
Faculty of Electrical Engineering and Computing

Publisher:

University of Zagreb, Faculty of Electrical Engineering and Computing

Organizers:

Bojana Dalbelo Bašić
Goran Glavaš
Mladen Karan
Jan Šnajder

Editors:

Goran Glavaš
Jan Šnajder

Publication Editor:

Jan Šnajder

ISBN 978-953-184-200-6

Powered by:

Text Analysis and Knowledge Engineering Lab
University of Zagreb
Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
http://takelab.fer.hr

TakeLab

Preface

With the advent of the digital age, the need to automatically analyze textual data has grown in importance, culminating in the emergence of the Big Data era we are experiencing nowadays. Driven in particular by the success of machine learning, research in natural language processing (NLP) and information retrieval (IR) has made significant progress over the last decades. At the same time, text analysis and retrieval applications have found their way to the market and have become a part of our everyday life. Acknowledging the importance of the field, University of Zagreb, Faculty of Electrical Engineering and Computing (UNIZG FER) introduced Text Analysis and Retrieval (TAR)[1] as a new course in its AY 2013/2014 curriculum. TAR brings together topics from NLP, IR, and relevant machine learning techniques, aiming to provide Computer Science master students with an understanding of the theoretical foundations and applications of text analysis and retrieval methods, as well as best practices, trends, and challenges in these dynamic fields.

This booklet presents the results of student projects done as part of TAR coursework. The AY 2013/2014 premiere edition of TAR generated a great deal of interest among the students, with as many as 48 students enrolled in the course. We've offered ten project topics,[2] nine of which were eventually bidded for. Out of 19 teams, 17 have submitted their project, and 16 submissions are included in this booklet (15 in English and two in Croatian). Obviously, we assigned some topics to more than one team; we felt that this may provide grounds for a better comparison and also foster a spirit of healthy competition.

We asked the students to write up their project reports in a form of a scientific article, based on the guidelines we provided. Not only that we believe that this contributes to the scientific and technical writing skills, but we also felt that this would ensure that the lessons learned stick long after the course is over. Moreover, we felt that this is a great way of passing on knowledge to others, most notably future generations of TAR students. This being said, we must note that the project results have not been reviewed or vetted. While we have of course provided guidance and feedback, we have not reviewed the final versions of the reports. As a consequence, the projects and reports are of varying quality. Most of the teams did a decent job, while some excelled and did much more than we've hoped for. Nonetheless, we believe that most project reports will be valuable for the readers, be it as a reference point, supplementary study material, or perhaps a source of inspiration.

We wish to thank all TAR 2014 students for their contributions, efforts, and enthusiasm.

Jan Šnajder and Bojana Dalbelo Bašić

[1] http://www.fer.unizg.hr/predmet/apt

[2] http://www.fer.unizg.hr/_download/repository/TAR-2014-ProjectTopics.pdf

Contents

Visualization of Large Text Collections in 2D

Stjepan Antivo Ivica, Frano Perleta

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{frano.perleta,stjepan-antivo.ivica}@fer.hr

Abstract

Large collections of text may be vizualized through dimensionality reduction while preserving semantic similarity between documents. We implement both Latent Semantic Analysis and Multidimensional Scaling, and apply those techniques to a collection of newsgroup messages.

1. Introduction

When faced with large amounts of data of any kind, it is often useful to represent it visually in order to leverage the innate human ability to recognize patterns and spatial relationships. This involves mapping the data into 2D or 3D space, in a way which preserves information meaningful for a particular application.

The goal of our project is visualization of large collections of textual documents. The primary objective is to capture the semantics of text in some way so that the visual representation of the collection reflects relationships between documents such as pertinence to related topics. There are many ways to extract semantic information from natural language, with various tradeoffs in terms of precision, computational feasibility, etc.

Our assignment is to apply the two methods of Latent Semantic Analysis and Multidimensional Scaling (henceforth refered to as LSA and MDS respectively) to this problem. We assume a very simple language model, representing documents with elements of a high-dimensional vector space. The problem can therefore be formulated as dimensionality reduction, and in principle any of the various techniques may be used, not necessarily specific to natural language processing.

Loosely speaking, the degree to which any two particular documents overlap in topic, ie. their semantic similarity, induces a certain topology on the document space. While this is a very informal idea, LSA and MDS are techniques intended to approximate that topology in a low-dimensional space.

We apply our implementation to the 20-newsgroups dataset[1], which is a large collection of newsgroup messages. By their very nature, individual messages are classified into 20 newsgroups, which are themselves grouped into 6 coarse classes. One would expect documents within a class to be close to each other, but there might also be considerable overlap between classes.

2. Related research

Large-scale Latent Semantic Indexing is discussed at length in (Zhang and Zhu, 2008). Although not directly related to visualization, the algorithm described there directly inspired our implementation of LSA.

[1] http://qwone.com/~jason/20Newsgroups/

3. Description

Our implementation consists of multiple standalone programs:

- `tar-lsa` is a command-line tool written in Haskell which performs dataset preprocessing and transformation using LSA.
- `mds.jar` is a command-line tool written in Java. It transforms a collection of vectors using MDS.
- `visual.jar` provides an interactive graphical interface which enables users to explore a 2D visualization of the dataset and retrieve individual documents.

These programs require that the dataset is available in two forms: as a collection of text files, each containing a single message; and as MATLAB®-compatible matrix files, containing among other things the word counts for all documents. Both versions of the dataset are available online.

3.1. LSA preprocessing

Due to the size of the dataset (19k documents and 61k terms), it is impractical, if not outright infeasible, to store the entire term-document matrix and calculate its singular value decomposition. The `tar-lsa` tool is used to determine a subset of documents and terms which will be used to construct a close approximation to the real term-document matrix.

The dataset is processed as a stream of document vectors, each thrown away immediately after updating the accumulators. When the entire pass through the dataset is complete, the magnitude of each document vector and the total number of occurences of each term are known, as well as the number of distinct documents containing each term.

- A fixed number of documents is selected, in order of decreasing ℓ^2-norm. This minimizes the approximation error with respect to the Frobenius norm (Zhang and Zhu, 2008).
- Terms are left out if they have too few occurences, since they are likely noise; or if they occur in too few distinct documents, since such terms contain very little "topological" information we are interested in.

- The identifiers of selected documents and terms are written to a pair of files to be read by subsequent invocations of `tar-lsa lsa`.

All the parameters for selection may be passed via command-line arguments – detailed instructions are available by invoking `tar-lsa prepare --help`.

3.2. LSA transformation

The transformation needs to be computed only once for a particular choice of output dimensionality. It proceeds as follows:

1. The identifiers of N documents and M terms which were previously selected are loaded from the filesystem. A dense $M \times N$ matrix $\mathbf{A}$ is allocated, and a pair of `IntMaps` is used to map sparse document and term identifiers into contiguous matrix indices.

2. A pass is made through the dataset, and selected documents are used to initialize the matrix $\mathbf{A}$. Subsequently, tf-idf weighting is applied to $\mathbf{A}$, with augmented term frequencies:

$$tf(t,d) = \frac{1}{2} + \frac{\mathbf{A}_{t,d}}{2 \max_\tau \mathbf{A}_{\tau,d}} \tag{1}$$

3. The thin singular value decomposition of $\mathbf{A}$ is computed:

$$\mathbf{A} = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top, \tag{2}$$

   where $k = \min\{M, N\}$ (which is N in practice), $\mathbf{U}_k$ is $M \times k$, $\mathbf{\Sigma}_k$ is $k \times k$, and $\mathbf{V}_k$ is $N \times k$. When seen as a linear operator, $\mathbf{A}$ is effectively factorized through a k-dimensional concept space. The columns of $\mathbf{U}_k$ represent concepts as basis vectors in the document space, while the numbers on the main diagonal of $\mathbf{\Sigma}_k$ signify the relative importance of each concept.

4. Another pass is made, in which each individual document vector (consisting of occurence counts) is first orthogonally projected onto the M-dimensional subspace, normalized, and finally mapped into the concept space:

$$\hat{\mathbf{d}} = \mathbf{\Sigma}_k^{-1} \mathbf{U}_k^\top \mathbf{d} \tag{3}$$

   If the output dimensionality is D, the final orthogonal projection is onto the subspace spanned by the vectors corresponding to D largest singular values. The output vectors are written to the filesystem.

The most computationally intensive part of the transformation is the SVD. We used the `hmatrix` library (Ruiz, 2014) for Haskell, which provides a purely functional interface to BLAS, LAPACK and GSL. The entire transformation takes on the order of tens of seconds, depending on M and N. The *thin* SVD algorithm provided by the `hmatrix` library avoids the costly and unnecessary calculation of singular vectors not corresponding to the largest k singular values.

For detailed usage instructions, run `tar-lsa lsa --help`.

3.3. MDS transformation

Whereas LSA uses a low-dimensional concept space to capture semantic similarity, MDS uses a dissimilarity measure on the document space and produces a mapping into a low-dimensional space which preserves that measure as well as possible. This method, as well as issues researches are confronted with when applying it, are discussed in (Wickelmaier, 2003). We used MDSJ, a free Java library which implements the algorithm described in (Brandes and Pich, 2007).

The computation of MDS proved to be memory consuming. To alleviate this problem, we first pass the document vectors through LSA in order to reduce the dimensionality to k. We experimented with various settings such as $k = 4$, $k = 1000$. The best way to calculate dissimilarity is Manhattan distance due to high similarity among the documents. Processing is done in batches of 200 documents to further scatter their representations in the final space.

3.4. Visualization

The visualization tool `visual.jar` is built in Java using the Prefuse library (Heer et al., 2005). Its arguments are: the path to a file containing triples of the form $(documentID, x, y)$; the path to a file containing numeric labels of each document; a flag that switches between fine or coarse classes; and path to the directory containing raw text documents. The graphical interface is interactive. The user can navigate using left clicks, zoom in and out using right clicks, and reset the view using double right clicks.

All documents are represented with shapes determined by coarse classes. Fine classes are distinguished using color. Information about a document is displayed when the user hovers the mouse cursor over the shape. Left clicking on the shape opens the corresponding text file in the directory provided as the last command-line argument.

4. Evaluation

Different visualizations of a subset of the document collection can be seen in figure 1.

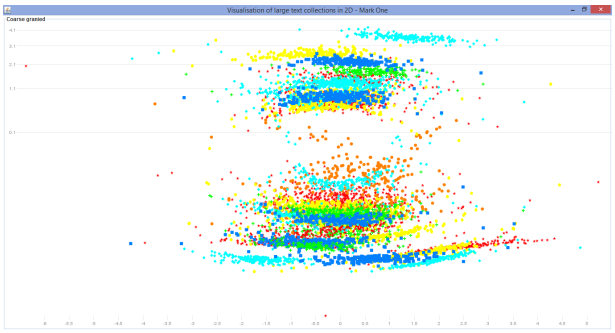
LSA by itself seems insufficient for this purpose. Although not entirely useless, the classes overlap almost completely, and almost no insight can be gained through visual inspection.

MDS yields somewhat more satisfying results. Although the contiguity of coarse classes has been lost, it is mostly preserved for fine classes.

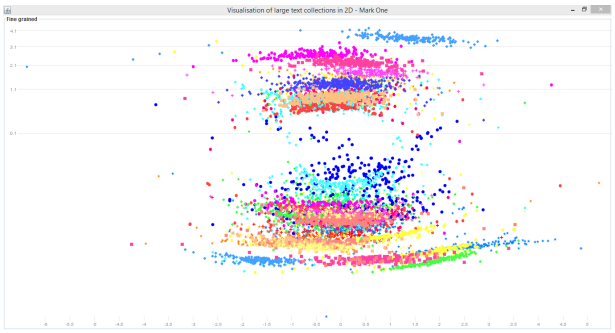
5. Conclusion

LSA and MDS are two relatively simple and well understood techniques applicable to text visualization. Both can be calculated efficiently. However, our experiments with these ended with mediocre results.

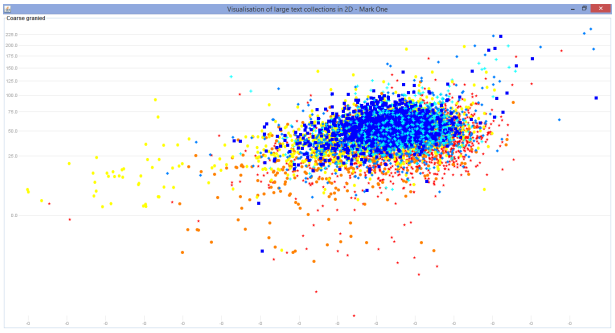
Although we are certain that a more rigorous implementation of these techniques would likely surpass ours in efficiency and usefulness, it seems worthwhile to explore other possible approaches to text collection visualization, such as deep learning.



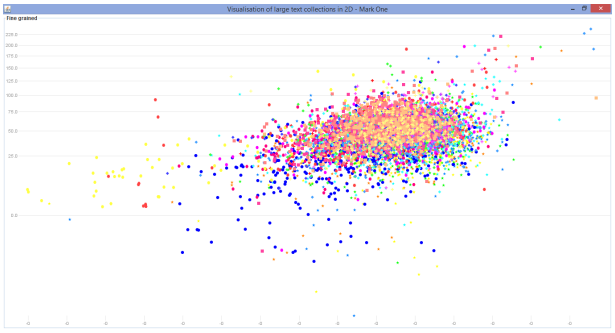
(a) MDS, coarse classes



(b) MDS, fine classes



(c) LSA, coarse classes



(d) LSA, fine classes

Figure 1: Visualization of a subset of the 20-newsgroups dataset with both algorithms and classification schemes.

References

Ulrik Brandes and Christian Pich. 2007. Eigensolver methods for progressive multidimensional scaling of large data. In *Graph Drawing*, pages 42–53. Springer.

Jeffrey Heer, Stuart K. Card, and James Landay. 2005. Prefuse: A toolkit for interactive information visualization. In *ACM Human Factors in Computing Systems (CHI)*, pages 421–430.

Alberto Ruiz. 2014. hmatrix: A haskell library for numerical computation. `https://github.com/albertoruiz/hmatrix`.

F. Wickelmaier. 2003. An introduction to MDS. *Reports from the Sound Quality Research Unit (SQRU)*, 7.

Dell Zhang and Zheng Zhu. 2008. A fast approximate algorithm for large-scale latent semantic indexing. In *Digital Information Management, 2008. ICDIM 2008. Third International Conference on*, pages 626–631. IEEE.

Link Analysis Algorithms (HITS and PageRank) in the Information Retrieval Context

Toni Antunović, Matija Delač

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
toni.antunovic@fer.hr, matija.delac@fer.hr

Abstract

This document contains description of information retrieval system that was created as part of Text Analysis and Retrieval course at Faculty of Electrical Engineering and Computing, Zagreb. We created a simple information retrieval system using a binary vector-space retrieval model and implemented link analysis algorithms PageRank and HITS. Performance of IR system was evaluated both with and without PageRank and HITS using Mean Average Precision.

1. Introduction

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers) (Manning et al., 2009). We've done this basic task using binary vector-space retrieval model. After that we refined the results using PageRank and HITS algorithms.

The analysis of hyperlinks and the graph structure of the Web has been instrumental in the development of web search. Study of link analysis builds on two intuitions(Manning et al., 2009)

- The anchor text pointing to page B is a good description of page B.
- The hyperlink from A to B represents an endorsement of page B, by the creator of page A.

The performance of the IR system both with and without PageRank and HITS was evaluated on the European Media Monitor test collection using standard IR metric Mean Average Precision.

2. The analysis of hyperlinks

PageRank and HITS are algorithms used to rank websites in search engine results. PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites (Page et al., 1999). HITS defines hubs and authorities. A page is considered a hub if it contains many hyperlinks, and an authority if many hyperlinks point to it. A page is a good hub if it points to good authorities, and a good authority if it is pointed to by good hubs (Bašić and Šnajder,).

2.1. The web as a graph

The Web is full of instances where the page does not provide an accurate description of itself. Thus, there is often a gap between the terms in a web page, and how web users would describe that web page. Consequently, web searchers need not use the terms in a page to query for it. In addition, many web pages are rich in graphics and images, and/or embed their text in these images; in such cases, the HTML parsing performed when crawling will not extract text that is useful for indexing these pages (Manning et al., 2009).

The fact that the anchors of many hyperlinks pointing to some page contain certain key words can be exploited by web search engines. To achieve that the web is represented in a form of a directed graph, where nods are web pages and connections between them are hyperlinks from one page to another (Kumar et al., 2000).

2.2. Test collection

We worked with European Media Monitor test collection which contains 1741 documents and a set of 50 queries with relevance judgments. Documents did not contain any hyperlinks so we used cosine similarity between their binary vector-space representations to determine if there is a link between them.

3. Preprocessing

Each document was represented using binary vector-space representation. This means that each document was represented as a vector with elements equal to 1 if it contains the corresponding index term, and 0 if it doesn't. Before that, stopwords removal, stemming using Porter stemmer and indexing was done.

4. Result retrieval

Retrieval of results was done using cosine similarity between documents and query and later refined with HITS or PageRank.

For HITS N best matches were retrieved, and then expanded with more documents that had measure of similarity greater than certain adjacency threshold. We set N to be 7, and adjacency threshold for case of expanding relevant document list to 0.55. While creating adjacency matrix L used by HITS, adjacency threshold of 0.47 was used. After that we let HITS converge using following update rule:

$$\mathbf{x}^{(\mathbf{k+1})} = L^T \mathbf{y}^{(\mathbf{k})}$$
$$\mathbf{y}^{(\mathbf{k+1})} = L \mathbf{x}^{(\mathbf{k})}$$

Table 1: MAP scores obtained

Basic IR	PageRank	HITS
0.158	0.162	0.18

where $\mathbf{x}$ is vector of authority scores for all documents and $\mathbf{y}$ vector of hub scores, both initialized to ones at the beginning. After every iteration vectors are normalized and algorithm is run until it converges or reaches 200 iterations. Final HITS score for each document was calculated as the sum of authority score and hub score and was used to rank previously determined relevant documents for the query.

PageRank used adjacency threshold of 0.35, and overall score $S^{(i)}$ for document $\mathbf{d^{(i)}}$ for query $\mathbf{q}$ was calculated as:

$$S^{(i)} = cos(\mathbf{q}, \mathbf{d^{(i)}}) + 5 \cdot \pi^{(i)}$$

π is a vector of PageRank scores for each document. It is calculated by applying following update on it until it converges:

$$\pi_{k+1}^T = \alpha \pi_k^T H + (\alpha \pi_k^T \mathbf{a} + 1 - \alpha) \frac{\mathbf{e}^T}{n}$$

where α is a scalar between 0 and 1, $\mathbf{a}$ is a binary vector and $a_i = 1$ only if document i has no hyperlinks. $\mathbf{e}$ is a vector of ones and n is number of documents.

5. Performance

Performance of IR system with and without HITS and PageRank was evaluated using Mean Average Precision. Results are shown in Table 1. HITS has the best score, but it is at the same time most computationaly intensive, that is after the initial convergence of PageRank scores is done at the beginning.

6. Conclusion

This was a simple project done as a part of a faculty course. There is a lot of room for improvement. Obvious idea would be using tf-idf weighting scheme instead of binary vector-space model and further adjusting parameters of algorithms to achieve better scores. Another possibility would be experimenting with alternative ways for representing documents and queries that would not be just bag-of-words representations.

References

Bojana Dalbelo Bašić and Jan Šnajder. *Improved Search, Evaluation, and Web Search*. Faculty of Electrical Engineering and Computing,University of Zagreb.

Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, Dandapani Sivakumar, Andrew Tompkins, and Eli Upfal. 2000. The web as a graph. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–10. ACM.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2009. *An Introduction to Information Retrieval*. Cambridge University Press.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The pagerank citation ranking: Bringing order to the web.

Authorship Attribution

Jelena Baksa, Marko Đomlija, Florijan Stamenković

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
jelena.baksa@fer.hr, florijan.stamenkovic@fer.com, marko.domlija@fer.hr

Abstract

This paper gives an overview of the system for authorship attribution which was built for the project in the Text Analysis and Retrieval class at FER. The main task of authorship attribution is assigning an author from the predefined author set to each text sample in the collection. Since the problem can be viewed as a text categorization task, our solution heavily relies on the supervised machine learning models. The system utilizes two types of classifiers: feature-based and compression-based. At the end, the classification performance was evaluated and compared in the terms of F-score.

1. Introduction

Authorship attribution is a method for assigning an author to each text sample in the data set (given the fixed set of authors). The whole concept is based on the assumption that we can differentiate between various text samples by measuring textual features of each sample (Stamatatos, 2009). Most commonly used authorship analysis methods heavily rely upon supervised machine learning, owing to the fact that the main idea of authorship attribution can be viewed as a text categorization task (Lewis and Ringuette, 1994).

The earliest efforts to solve authorship attribution problem come from the late 19th century, when Mendenhall performed statistical studies on the plays of Shakespeare (Williams, 1975). One of the most important works in this field comes from the middle of the last century: an extensive study by Mosteller and Wallace on series of 146 political papers called "The Federalist Papers" (Mosteller and Wallace, 1964). The research which followed up to the end of 20th century was dominated by stylometry research. Since then, a lot of factors with immediate impact on authorship attribution have changed; namely availability of electronic texts on the Internet, more efficient computation techniques and more powerful algorithms were designed for various NLP tasks. For this reason, the last decade can be marked as a new phase in authorship attribution research (Stamatatos, 2009).

Authorship attribution can be applied to the tasks such as plagiarism detection or author verification, and in various areas such as criminal and civil law, computer forensics or intelligence (counter-terrorism).

2. Problem definition

Our task is to perform and evaluate authorship attribution on an ad-hoc authorship attribution competition dataset by Patric Juola[1]. The dataset consists of ten labeled plays from three English Renaissance authors (Ben Jonson, Christopher Marlowe and William Shakespeare). Four plays by Jonson are in the set, three by Marlowe and four by Shakespeare.

We will attempt to define a classification method for an unseen piece of text. We will utilize common techniques in authorship attribution, most of which are based around supervised machine learning methods. Finally, we will evaluate obtained results using standard evaluation measures.

There are some observations to be made on the dataset we will be working with. It contains texts by authors from the same language area and timeframe. The texts belong to the same genre of fictional literature. In that sense our task is non-trivial. On the other hand, the training set of texts is plentiful and we are to distinguish among only three authors. In many real world authorship attribution problems this will not be the case. Therefore, any results we may obtain must be viewed in context of this particular dataset.

2.1. Dataset organization

Each play contains between two and six thousand lines (4270 on average) and between 16 and 40 thousand words (28 thousand on average). In order to get statistically justifiable evaluation results, each play was split into smaller fragments of text. Plays were split into fragments of desired word, line or character length. Splitting always occurred on line endings (note that this does not necessarily coincide with sentence endings, thus sentencing across fragments was affected). Since the process is automated, it was possible to evaluate classifier performance for various resulting play fragment lengths.

3. Classifiers

In accordance to existing work in the area of authorship attribution, various approaches were tried when looking for the best solution to the problem. Following is an overview of these approaches with a brief description of each.

3.1. Feature based classifiers

A number of well known classifiers that are based on numeric features extracted from text were evaluated. These are:

- naive Bayes (NB) classifiers
- support vector machines (SVM)
- logistic regression
- k-nearest-neighbors (KNN) algorithms

[1] http://www.mathcs.duq.edu/~juola/problems/problemD

For naive Bayes methods three approaches were tried: Bernoulli, multinomial and Gaussian. A Bernoulli NB classifier is based on feature occurrence probability within a class, for example the occurrence of a word or a character n-gram in text. A multinomial NB classifier is based on the frequency with which a feature occurs within a class. Both of these kinds of classifiers are therefore not applicable to features not based on occurrences, such as for example sentence length. When training Bernoulli and multinomial NB models, minimal additive smoothing ($\alpha = 10^{-9}$) was used only to avoid numeric problems. Gaussian NB classifiers on the other hand estimate a Gaussian distribution of each numeric feature within a class. Thus they are applicable to features not based on occurrences, but are not applicable to non-numeric features. Since only numeric features were extracted from text, Gaussian NB models were applicable to all feature sets.

Support vector machines were used only with a linear kernel. Like with Gaussian NB, SVMs can work with numeric features and were applicable to all feature sets. Data pre-processing was done to ensure unit variance for each feature across the whole feature set. Two different approaches to linear class separation were tried, the 1-vs-1 approach and 1-vs-rest approach.

k-nearest-neighbors classifiers were used with the k parameter ranging from 3 to 13. These classifiers are applicable to any numeric feature set.

For all classifiers the scikit-learn[2] library was used, an open-source Python based machine-learning library available under the BSD license.

3.2. Feature extraction

Various feature types were considered during our analysis of the previous studies on authorship attribution. The features which are most commonly used in such tasks can be categorized to the following groups (Stamatatos, 2009):

- lexical
- character
- syntactic
- semantic
- application-specific

The criteria used for deciding which features will be implemented as a part of this projects were computational complexity and overall importance of each feature, according to Stamatatos (2009). The goal was to choose different feature types and compare them by standard evaluation metrics, such as F-score, Precision and Recall. At the end, we decided to implement the following feature types:

- character frequency
- character-based n-grams frequency
- word frequency
- stop word frequency
- word-based n-grams frequency
- part-of-speech (POS) tag n-grams frequency

[2] `http://scikit-learn.org`

Extraction of the character features is one of the simplest methods because it does not require any additional tools (such as tokenizer used in word extraction). In addition to the basic features such as character count, a feature that was also considered were character-based n-grams. This approach can be quite successful, since character n-grams manage to capture lexical and contextual information. It is also suited for the languages where word extraction and tokenization task is too hard (Matsuura and Kanada, 2000).

Word frequency feature views the text as a simple set of words, excluding the contextual and semantic information. Additionally, some studies (Gamon, 2004) have found that the words that are usually not considered in text-classification methods (stop words) are often a very good way to distinguish between the authors. On the other hand, word n-grams are a good way to take into the account contextual information from the text. However, word-based features have the disadvantage of often capturing content-specific information rather than stylistic information.

Since syntactic information can often be more reliable than lexical information for authorship attribution tasks, another feature which was considered in this project were n-grams of part-of-speech (POS) tokens. This approach is interesting because it is one of the simplest forms of the syntactic-based text representation. The only requirement for the computation of POS tag n-grams is availability of the tool for POS tagging. Even though Stamatatos (2009) mentioned some more sophisticated methods in this area, such as rewrite rules for phrases in the sentences, this method was chosen mostly for its simplicity.

In order to perform some common NLP tasks (such as stemming or POS tagging), several NLP libraries were considered, and two were used during feature extraction, namely Stanford CoreNLP[3] and Apache OpenNLP[4]. In our final implementation we used only Apache OpenNLP, since the overall performance and speed was quite better than the Stanford library.

3.3. Parameter optimization

Even though some of the classifiers have a hyper-parameter to be optimized (SVM, KNN), no formal hyper-parameter optimization was performed. There are two reasons for this. Firstly, informal optimization (observing classifier performance for different hyper-parameter values without using a separate validation dataset) has shown that using many different hyper-parameter values made almost no effect on classifier performance. For selected feature sets all SVMs gave virtually the same measured performance for the hyper-parameter 'c' ranging from 0.01 to 10^6. This can be explained by observing that having $c = \infty$ results in a hard-margin SVM, which should still perform well at linear separation of three classes in input space of 50 to 300 dimensions. The k parameter in KNN classifiers also gave virtually identical results for k having all odd values between 3 and 21.

The second reason for not using formal hyper-parameter optimization is that the data set consists of only ten plays.

[3] `http://nlp.stanford.edu/software/corenlp.shtml`

[4] `https://opennlp.apache.org/`

When evaluating a classifier performance for a text fragment, the classifier training set must exclude all fragments from the same play. If then also a validation set was used for hyper-parameter optimization, another play would have to be excluded, resulting in a total of only eight plays used for a three author classification problem.

3.4. Compression based classifiers

The second type of classifier is a compression based one. In this approach text similarity is measure using an off-the-shelf compression algorithm (Benedetto et al., 2002). A brief description of this approach follows. Firstly, all texts from a particular author are taken and concatenated to a file denoted W_i. Then W_i is compressed into a binary file C_i. The training phase consists of repeating this process for all authors. When examining a particular test sample T_x, it is appended to a W_i and compressed into N_{xi}. Finally, the difference between the size of N_{xi} and C_i is examined. Sample T_x is attributed to the author whose compressed file size increases the least when the sample is appended to it.

The main idea behind this method is to examine how much the size of a compressed file increases when a particular text fragment is appended to it. The file is first compressed without the fragment, and then with the fragment appended. The more the fragment is similar to the original file, the less it will affect the file size. Compression is done using the ZLIB.NET[5] open source library.

Compression method used applies *sliding window* compression to the file. Basically, the algorithm finds recurring sequences of characters and replaces the recurring sequence with a reference to its instance that appeared earlier in the text. In the example given above, the more the appended file fragment is similar to the original file (more sequences from the appended fragment that are found in the original text), the better the compression. Thus the fragment is attributed to the author whose original text corpus it is most similar to.

4. Results

Following are the results of extensive classifier evaluation performed. Among classifier types and their variants, four that have shown representative (typical, not best) performance in the group are presented:

- multinomial naive Bayes classifier
- 5-nearest-neighbors
- linear 1-vs-1 SVM with hyper-parameter 'c' set to 1
- compression (ZIP) based classifier

The feature-sets used for evaluation are based on frequencies of:

- individual characters
- top 300 (most frequent) character trigrams
- top 100 (most frequent) word stems
- stop-words
- part-of-speech tags

[5] `http://zlibnet.codeplex.com/`

Table 1: Classifier evaluation results (F1 scores) for 1401 fragments of c. 200 words.

Feature set	N. Bayes	5-NN	SVM	ZIP
Character freq.	0.67	0.54	0.63	0.86
Top 300 char-trigrams	0.85	0.73	0.88	0.86
Top 100 words	0.62	0.48	0.53	0.86
Stop-words	0.57	0.42	0.42	0.86
POS tags	0.55	0.38	0.42	0.86

Table 2: Classifier evaluation results (F1 scores) for 281 fragments of c. 1000 words.

Feature set	N. Bayes	5-NN	SVM	ZIP
Character freq.	0.80	0.68	0.72	0.80
Top 300 char-trigrams	0.90	0.80	0.92	0.80
Top 100 words	0.67	0.72	0.66	0.80
Stop-words	0.64	0.53	0.45	0.80
POS tags	0.67	0.47	0.59	0.80

Each feature-set was evaluated separately. Our aim was to determine how well different types of features perform. Naturally, it would be possible to extract the most discriminant features of each group and combine them in a single classifier. It is to be expected that such an approach would further improve classifier performance, but is beyond the scope of this paper.

Evaluation scores are collected using a modified leave-one-out cross validation. The method is modified to ensure that classifiers are not trained on samples originating from the same play as the sample currently being evaluated. Evaluations presented are the F1 macro scores.

Evaluation results are shown in three tables. Each table shows the F1 macro scores for all classifiers and feature-sets, but different sizes of text fragments of the original plays. Also results for the compression based classifier are provided. Since this classifier is not based on extracted features, its result is the same for all rows in a single table. Table 1 shows the results for plays being split into units of text in average 200 words long, table 2 for plays split into units of 1000 words and table 3 for plays split into units of 2500 words.

From the evaluation results presented in score tables it is possible to draw several conclusions. Compression based classifiers seem to perform quite well. This is not very surprising if we consider the principles of compression employed in the process, but is a very unorthodox approach from the supervised machine learning standpoint. Regarding more conventional, feature-based classifiers, multinomial naive Bayes and the linear SVM classifiers give generally very similar results, naive Bayes often showing slightly better performance. Greater performance disproportions can be seen when comparing results for different feature-

Table 3: Classifier evaluation results (F1 scores) for 112 fragments of c. 2500 words.

Feature set	N. Bayes	5-NN	SVM	ZIP
Character freq.	0.91	0.79	0.82	0.86
Top 300 char-trigrams	0.94	0.85	0.94	0.86
Top 100 words	0.67	0.75	0.73	0.86
Stop-words	0.68	0.65	0.56	0.86
POS tags	0.74	0.58	0.66	0.86

sets. It is apparent that in all three text fragmentations classifiers based on character trigrams perform best. This is in accordance with the existing work in the field (Sanderson and Guenter, 2006). Character frequency also performs well. Word frequency gives decent results, but should be used with caution due to the thematic content it carries. Contrary to our expectations, stop-word and POS tag frequency do not perform well. It was assumed that these features would carry a lot of style content and would be well suited to authorship attribution tasks, but that does not seem the case. Finally, it seems that all classifiers on all feature-sets perform better on longer fragments of text. This is in line with an intuition that more text carries more information, and results in less feature variance within a class.

Finally, it must be noted that the overall best result (0.94 F1 score for character trigrams on fragments of 2500 words) is suspiciously good, and indicates a potential fault in our method of evaluation. We have been unable to locate such an error, but remain reserved concerning these results.

5. Conclusion

Authorship attribution can be viewed as simple text categorization task. As a consequence, the system built for the purpose of this assignment heavily relied upon supervised machine learning methods. Several different methods for solving the authorship problems were researched, analysed and implemented.

Feature-based classifiers are a well-known method for authorship analysis. In order to learn what kind of settings work best for this approach, we evaluated four classifier types (naive Bayes, support vector machines, logistic regression, k-nearest-neighbors algorithms) across six feature sets (character/word frequency, character/word n-gram frequency, stop word frequency, POS tag n-gram frequency). These classifiers were applied to three different dataset fragmentations (200/1000/2500 word fragments). Another approach which we implemented were compression-based classifiers.

Classification evaluation confirmed the findings from previous studies. Best evaluation results were obtained for character trigram features, thus confirming our initial assumptions. On the other hand, evaluation results for stop-word and POS-tag based features were worse than we expected. Finally, even though compression-based classifiers are a somewhat unconventional approach to authorship attribution problem, they seem to perform quite well.

References

Dario Benedetto, Emanuele Caglioti, and Vittorio Loreto. 2002. Language trees and zipping. *Phys. Rev. Lett.*, 88:048702, Jan.

Michael Gamon. 2004. Linguistic correlates of style: authorship classification with deep linguistic analysis features. In *Proceedings of the 20th international conference on Computational Linguistics*, page 611. Association for Computational Linguistics.

David D Lewis and Marc Ringuette. 1994. A comparison of two learning algorithms for text categorization. In *Third annual symposium on document analysis and information retrieval*, volume 33, pages 81–93.

Tsukasa Matsuura and Yasumasa Kanada. 2000. Extraction of authors' characteristics from japanese modern sentences via n-gram distribution. In *Discovery Science*, pages 315–319. Springer.

Frederick Mosteller and David Wallace. 1964. Inference and disputed authorship: The federalist.

Conrad Sanderson and Simon Guenter. 2006. Short text authorship attribution via sequence kernels, markov chains and author unmasking: An investigation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 482–491. Association for Computational Linguistics.

Efstathios Stamatatos. 2009. A survey of modern authorship attribution methods. *Journal of the American Society for information Science and Technology*, 60(3):538–556.

Carrington B Williams. 1975. Mendenhall's studies of word-length distribution in the works of shakespeare and bacon. *Biometrika*, 62(1):207–212.

Named entity recognition in Croatian tweets

Baksa Krešimir, Dino Dolović

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{kresimir.baksa, dino.dolovic}@fer.hr

Abstract

This paper presents our work done in named entity recognition (NER) for tweets in Croatian and Serbian. We have used Hidden Markov Models (HMM) and Conditional Random Fields (CRF) as models for automated labeling of words. Finding named entities in tweets such as person names, organizations and locations is a much harder task than extracting them from regular document due to some cons found in tweets which are discussed more briefly in this paper.

1. Introduction

Named-entity recognition (NER) (also known as entity identification, entity chunking and entity extraction) is a subtask of information extraction that seeks to locate and classify elements in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc (Wikipedia, 2014). There are three common ways that are mostly used in recognizing entities in documents: rule-based methods, machine learning methods and hybrid methods. As data grows, machine learning methods are much more efficient in this task then the others because the data can speak for itself and there is no need to design or maintain rules as in rule-based methods. The only problem is that the data must be annotated, or at least partially annotated for some semi-supervised methods.
Social media, like facebook or twitter, are new and not yet entirely explored. But there is a lot of useful information that could be used in real life. For example, measuring the person's or organization's popularity by finding the total number of their mentionings in different tweets.
NER task for documents is quite different than in tweets. Here are some properties that are found in tweets but not in documents:

- Tweets are only 140 characters long
- Usually tweets aren't grammatically and syntactically correct
- Some twitterers tend to write meaningless tweets
- Person names and organizations aren't always capitalized
- Names of some locations are not always properly written or are shortened

Those properties cause many problems in selecting the features for a model, because one cannot use as many and as good features as it would in NER for regular documents (Finin et al., 2010).
As shown in some studies (Ratinov and Roth, 2009), the performance of a NER system drops quite a lot when extracting named entities in tweets. For example, the average F1 of the Stanford NER (Finkel et al., 2005), which is trained on the CoNLL03 shared task data set (non-tweet data) and achieves state-of-the-art performance on that task, drops from 90.8% (Ratinov and Roth, 2009) to 45.8% when tested on tweets (Liu et al., 2011)
.

2. Methods

2.1. Tweets annotation

For this project we were given data from our TAs, that consisted of tweets in Serbian and Croatian. Although we extracted only Croatian tweets given the language attribute in XML tag of a tweet, we also got many Serbian tweets (roughly 60:40 in favor of Serbian) so we decided to work with mixed set. We performed automatic annotation on words in tweets which appeared in our dictionaries with person, oraganization and location names. Dictionaries with person and location names were downloaded from the web (*http://www.croatian-genealogy.com* for person names and *http://www.uprava.hr* for locations). Dictionary for organizations was created manually. Afterwards, we set the ground rules for annotating the data. Those rules are following:

- Each token is annotated separately as a person name [PER], organization [ORG], location [LOC] or nothing [NONE]
- PER - annotate names, surnames, nicknames, fantasy creatures but not their titles (eg. doc. dr. sc.) ...
- ORG - annotate names of organizations, institutions, state authorities, sport clubs, national sport teams, league sport names but not generic names like government, party etc.
- LOC - annotate places, regions, states, rivers, mountains, squares, streets ...
- do not annotate tokens starting with "@"
- annotate words written in "#"
- annotate words according to the tweet context
- when in doubt, whether to annotate the word as a location or organization, prefer organization

After establishing the rules, we manually went over all tweets, about 5k of them, and corrected the annotations that were incorrectly made by program or were missed. The annotation was done just by two of us. Some of the tweets, about 1k of them were labeled from both of us so we could measure *IAA* score, which was very high, about 98% agreement. The main reason for such a high score is because annotating tweets is quite easy and straightforward and mistakes are usually made in lack of concentration.

2.2. HMM

Hidden Markov Model (HMM) is powerful statistical tool for sequence labeling tasks. It is formed as an extension of Markov process where each state has all observations joined by probability of current state generating observation. The formal definition of a HMM is as follows:

$$\lambda = \big(A, B, \pi\big) \tag{1}$$

S is state set, and V is the observation set:

$$S = \{s_1, s_2, \ldots, s_N\} \tag{2}$$

$$V = \{v_1, v_2, \ldots, v_M\} \tag{3}$$

We define Q to be a fixed state sequence of length T, and corresponding observation O:

$$Q = q_1, q_2, q_3, \ldots, q_T \tag{4}$$

$$O = o_1, o_2, o_3, \ldots, o_T \tag{5}$$

A is transition matrix, storig the probability of state j following state i.

$$A = \big[a_{ij}\big], a_{ij} = P\big(q_t = s_j | q_{(t-1)} = s_i\big) \tag{6}$$

B is the observation array, storing the probability of observation k being produced from the state j:

$$B = \big[b_i(k)\big], b_i(k) = P\big(x_t = v_t | q_t = s_i\big) \tag{7}$$

π is the initial probability array:

$$\pi = [\pi_i], \pi_i = P(q_1 = s_i) \tag{8}$$

Given HMM, and a sequence of observations, we are able to compute $P(O|\lambda)$, the probability of the observation sequence given a model, and with that find out witch is the most probable state sequence producing given sequence of observations. For calculating probabilities we use Viterbi's algorithm. First we define:

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P(q_1 q_2 \ldots q_t = s_i, o_1, o_2, \ldots, o_t | \lambda) \tag{9}$$

as the probability of the most probable state path for the partial observation sequence.
The Viterbi algotirthm is as follows:

1. Initialisation:

$$\delta_1(i) = \pi_i b_i(o_1), 1 \leq i \leq N, \psi_1(i) = 0 \tag{10}$$

2. Recursion:

$$\delta_t(j) = \max_{1 \leq i \leq N} \big[\delta_{t-1}(i) a_{ij}\big] b_j(o_t), 2 \leq t \leq T, 1 \leq j \leq N \tag{11}$$

$$\psi_t(j) = arg \max_{1 \leq i \leq N} \big[\delta_{t-1}(i) a_{ij}\big], 2 \leq t \leq T, 1 \leq j \leq N \tag{12}$$

3. Termination:

$$P^* = \max_{1 \leq i \leq N} \big[\delta_T(i)\big] \tag{13}$$

$$q_T^* = arg \max_{1 \leq i \leq N} \big[\delta_T(i)\big] \tag{14}$$

4. Optimal backtracking:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), t = T-1, T-2, \ldots, 1 \tag{15}$$

The backtracking allows the best state sequence to be found from the back pointers stored in the recursion step.

2.3. CRF

Conditional Random Fields (CRF) are discriminative graphical models that can model overlapping, non-independent features in sequence of data. A special case, linear-chain CRF, can be thought of as the *undirected graphical model* version of HMM. Along a different dimension, HMMs are the sequence version of Naive Bayes models, while linear-chain CRFs are the sequence version of logistic regression.
Let $x_{1:N}$ be the observations (e.g., words in a document), and $y_{1:N}$ the hidden labels (e.g., tags). A linear chain CRF defines a *conditional probability* (whereas HMM defines the joint):

$$p(y_{1:N}|x_{1:N}) = \frac{1}{Z} exp\Big(\sum_{n=1}^{N} \sum_{i=1}^{F} \lambda_i f_i(y_{n-1}, y_n, x_{1:N}, n)\Big) \tag{16}$$

The scalar Z is a normalization factor, or *partition function*, to make $p(y_{1:N}|x_{1:N})$ a valid probability over label sequences. Z is defined as:

$$Z = \sum_{y_{1:N}} exp\Big(\sum_{n=1}^{N} \sum_{i=1}^{F} \lambda_i f_i(y_{n-1}, y_n, x_{1:N}, n)\Big) \tag{17}$$

Within the exp() function, we sum over $n = 1, \ldots, N$ word positions in the sequence. For each position, we sum over $i = 1, \ldots, F$ *weighted features*. The scalar λ_i is the weight for feature $f_i(\cdot)$. The λ_i's are the *parameters* of the CRF model and must be learned.
The feature functions are the key components of CRF. In case of linear-chain CRF, the general form of a feature function is $f_i(y_{n-1}, y_n, x_{1:N}, n)$, which looks at a pair of adjacent states y_{n-1}, y_n, the whole input sequence $x_{1:N}$, and where we are in the sequence. The feature functions produce a real value for example, we can define a simple feature function which produces binary values such as:

$$f_1(\cdot) = \begin{cases} 1 & if\ y_{n-1} = NONE\ and\ y_n = PER \\ 0 & otherwise \end{cases} \tag{18}$$

In our work, we used crf-suite, a CRF implementation written in C++ by (Okazaki, 2007). CRFsuite internally generates features from attributes in a data set. In general, this is the most important process for machine-learning approaches because a feature design greatly affects the labeling accuracy. In this work, we extract some features listed below:

- Token content
- Token shape (e.g. Dinamo has shape ULLLLL)
- Token type (e.g. InitUpper, AllUpper, ...)
- Prefix and suffix characters
- Multiple digit combination (e.g 1999.)
- All letter or digit flag
- Previous and next token content (up to range of four tokens)

3. Evaluation

We measured the performance of two models: HMM and CRF. We also developed two basline models, B1 and B2 in order to compare our models to some lower boundery models. B1 baseline model is a simple model that tags the words in a tweet randomly depending on the probability of a tag that was calculated from training set. For instance, if the calculated probability of a tag PER is 0.7, the model will generate tag PER for a word with the probability of 0.7 and will not consider any features of a word at all. Second basline model B2, will generate tag PER, LOC or ORG for those words that appear in our dictionaries, gazetteers. Other words that are not found in a dictionary will be tagged as NONE. In Table 1 we report the performance of precision, recall and F1 score for each named entity and each model.

It is clear that CRF model outperforms all other models achieving F1 score of 86.26. The reason that CRF is so better then rest of the models is because it includes various different features of a word and its context when generating a tag for a word. Features that are used in CRF model were explained in previous chapter.

From results we observed that HMM model did not beat baseline B2. HMM achieved only 5.54 F1 score for tagging locations when B2's F1 score is six times larger. This is because there weren't many locations present in training set (nor in the rest of the hand labeled tweets) so the HMM model learned the pattern of recognizing locations very poorly. On the other hand, B2, which only uses gazzeters in order to generate a tag for a word, was better at this task because it classified the word depending on if the word was a gezzeter or not. CRF F1 score for location goes even higher, up to 67.35. This is because the CRF uses features of several previous and next words in order to generate a tag for a current word and those features tend to be very important in Croatian. For instance, when one mentions a location in a tweet, it is quite probable that previous word will be of a small lenght like *u, na, iz* and CRF will use this information when classifying a word. CRF also outperforms other models when recognizing organiztaions. Like locations, organizations in tweets also follow some patterns. For instance, organizations in Croatian are likely to be consisted of a multiple words like *Ministarstvo znanosti, obrazovanja i športa* and CRF will also learn that as well very good.

4. Conclusion

In this paper we have presented two models for named entity recognition in tweets, HMM and CRF. We manually labeled the provided tweets, trained our models on training set and evaluated both models on the training set and got promising results with CRF's F1 score of 86.26. Due to problems encounterd in tweets and their nonformal nature, we think that the results are promising and could improve even more.

In future, we propose working with larger sets of annotated tweets. Furthermore, we propose annotating some other named entities such as events because they appear quite often in tweets and valuable knowledge could be extracted from that.

References

Tim Finin, Will Murnane, Anand Karandikar, Nicholas Keller, Justin Martineau, and Mark Dredze. 2010. Annotating named entities in twitter data with crowdsourcing. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, pages 80–88. Association for Computational Linguistics.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics.

Xiaohua Liu, Shaodian Zhang, Furu Wei, and Ming Zhou. 2011. Recognizing named entities in tweets. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 359–367. Association for Computational Linguistics.

Naoaki Okazaki. 2007. Crfsuite: a fast implementation of conditional random fields (crfs).

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics.

Wikipedia. 2014. Named-entity recognition — wikipedia, the free encyclopedia. [Online; accessed 7-June-2014].

	B1			B2			HMM			CRF		
NE class	P	R	F1	P	R	F1	P	R	F1	P	R	F1
PER	18.03	18.37	18.56	64.74	98.96	78.27	67.04	86.11	75.39	93.87	91.71	92.78
ORG	6.51	6.57	5.83	20.94	75.72	32.81	17.33	85.64	28.83	81.66	66.93	73.57
LOC	1.82	1.75	1.98	22.13	57.43	31.95	2.86	84.61	5.54	77.36	59.64	67.35
Overall Marco	8.63	8.76	8.69	35.94	77.37	47.68	29.08	85.45	36.58	84.29	72.76	78.10
Overall Micro	13.59	13.79	13.69	51.30	**93.82**	66.33	50.43	86.06	63.60	**90.27**	83.42	**86.26**

Table 1: Evaluation results

Text Classification and Evaluation

Ivan Borko, Sofia Čolaković, Ivan Stražičić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{ivan.borko,sofia.colakovic,ivan.strazicic}@fer.hr

Abstract

Machine learning approach to text classification has proven to give good results in numerous articles. In this paper we used Vector Space Model and feature selection to derive features that will be fed to several different classifiers. Features were selected from 50, 100, 500, 1000 and 5000 most informative words using Mutual Information measure. Documents were then represented using Vector Space Model with tf-idf weighting. Classification models evaluated in this paper were Naive Bayes, SVM, Logistic Regression and K-nearest Neighbours. Models were evaluated on 20 Newsgroups dataset. The best result of 90,5% F1 score was obtained using Logistic Regression classifier. Results shown in this paper are comparable to results presented in other papers using the same dataset.

1. Introduction

Text classification is a problem of assigning predefined classes or topics to text documents. Big problem in text classification is a very big feature space. This problem emerges because a feature is usually the word count of a single word in a document. This way number of features is always significantly larger than the number of examples in a collection. This can be the reason of failure for some machine learning algorithms.

Furthermore, there are many words that are too common or too rare to carry the information about the topic of a document. For reasons stated above, feature selection is a technique commonly used in text classification. In this paper Mutual Information measure was used for deriving the most informative words for every class. Mutual Information is a measure that defines mutual dependence between two variables, in this case between a class and a word.

After selecting features, documents have to be presented in a format that can be fed to the classifier. We used Vector Space Model with tf-idf weighting for document representation. This representation is most commonly used in text classification.

The last step in text classification is determining a topic using machine learning model. In this paper, two problems are presented, classification in one of 6 more general topics (politics, religion, science, ads, sports and computers) and classification in 20 broader topics. Classification in 20 topics is more difficult because some classes are alike and probably have similar word counts (e.g. comp.sys.ibm.pc.hardware and comp.sys.mac.hardware). We classified documents in multiple classes using multiple binary classifiers, with both *one-vs-one* and *one-vs-all* schemes.

2. Related Work

Large number of machine learning models can be applied in text classification. Probabilistic models like Naive Bayes are reported to work well on this problem despite the large feature space (McCallum and Nigam, 1998). Also, SVM is a method that shows superiority in some experiments (Joachims, 1998). Most commonly SVM with linear kernel is used but polynomial and RBF kernels show good results (Joachims, 1998).

Feature selection is the other problem where multiple methods can be applied. Most commonly used measures are Mutual Information and Chi Square, but there are measure that produce better result. In (Forman, 2003) an extensive study of feature selection metric is conducted. Best results are produced using Bi-Normal Separation (BNS).

Third aspect of text classification that can be improved is a document representation. In (Soucy and Mineau, 2005) weighting based on confidence and supervised weighting were proposed. Best results are achieved with weighting based on confidence, but supervised weighting also gives better results than tf-idf.

3. Used Dataset

Dataset used in this project is the 20 Newsgroups dataset which can be downloaded here `http://qwone.com/~jason/20Newsgroups/`. This data set has become popular choice for text related applications of machine learning, especially text classification and text clustering. It is divided on train set containing 11269 documents and 53975 words and test set containing 7505 documents and 47376 words. Document is classified as one of 20 topics which are organized into 6 topic groups. For each document word counts and classification is provided. Topics grouping in the dataset is shown on figure 1.

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

Table 1: Topic groups

4. Feature Selection and Data Preparation

On word counts provided in 20 Newsgroups dataset feature selection is applied. We used mutual information measure to derive words that are most informative for each topic. We

extract a group of 50, 100, 500, 1000 and 5000 words for each topic and take only the union of those words in consideration in further steps. Also, dataset was classified without feature selection for precision comparison. Selected features are then translated into vector space model using tf-idf weighting.

Mutual Information is used with smoothing to prevent unexpected behaviour when there are no words in documents of a certain classification. Mutual Information is calculated using the expression:

I(U;C) =

$$\frac{N_{11}}{N}\log_2\frac{NN_{11}+1}{(N_{1.})*(N_{.1})+1}+$$
$$\frac{N_{01}}{N}\log_2\frac{NN_{01}+1}{(N_{0.})*(N_{.1})+1}+$$
$$\frac{N_{10}}{N}\log_2\frac{NN_{10}+1}{(N_{1.})*(N_{.0})+1}+$$
$$\frac{N_{00}}{N}\log_2\frac{NN_{00}+1}{(N_{0.})*(N_{.0})+1}$$

N_{xy} -number of documents with the following attributes $x, y \in \{0, 1\}$

$$x = \begin{cases} 1 & \text{if document is classified in topic C} \\ 0 & \text{otherwise} \end{cases}$$

$$y = \begin{cases} 1 & \text{if word is in a document} \\ 0 & \text{otherwise} \end{cases}$$

Tf-idf is a statistical measure that reflects how important is a word for a document. It consists of two influences, one increases with word frequency in a document and the other decreases with word appearing in many documents. They are tf and idf measure and their product is the tf-idf measures. Tf-idf measure is calculated for every word that appears in a document. Also, the word has to be selected by Mutual Information in previous step to be acknowledged here.

$$tf(t,d) = 0,5 + \frac{0,5 * f(t,d)}{max\{f(w,d) : w \in d\}}$$

$$idf(t,D) = \log\frac{N}{|\{d \in D : t \in d\}|}$$

$$tfidf(t,d,D) = tf(t,d) * idf(t,D)$$

$tf(t,d)$ – measures frequency of a word t in a document d
$idf(t,D)$ – measures number of documents containing word t

5. Classification

After we managed to create the features vectors for all our documents, we had to classify them. First of all, we wanted to test different classifiers. Most of our tests were done on two types of datasets: first dataset is created from similar newsgroups merged into one class per topic, forming 6 classes (politics, religion, science, ads, sports and computers) and second dataset that is the original dataset of 20 newsgroups (20 classes).

Classifier	6 classes	20 classes
Logistic Regression	0.904	0.826
LinearSVM (liblinear)	0.904	0.820
LinearSVM (libsvm)	0.889	-
Naive Bayes, Multinomial	0.897	0.815
Naive Bayes, Bernoulli	0.823	0.750
Nearest Neighbours	0.151	-

Table 2: Classifiers

Results are shown in table 2. Logistic Regression proved to be the best in our case. Input vectors were tf-idf features as described in previous chapter. We tested two versions of SVM with linear kernel. One was from liblinear package and it uses *one-vs-others* multiclass scheme. Other was from libsvm package and it uses *one-vs-one* scheme. *one-vs-others* scheme proved to be better, so we didn't test the libsvm version on the 20 class dataset.

We also tried classification with SVM with RBF kernel, but it failed terribly. Training was very slow and F1 score results were low. This behaviour can be explained by looking at the input space dimensionality. We have around 11 000 documents in train set and the dimensionality of the input space (tf-idfs of words) is around 62 000. This means that the dimensionality of input space is much greater that the number of samples. Linear models work great with that kind of problems, but SVM with RBF fails. SVM with RBF tries to even increase the input space which is not really useful.

Nearest Neighbours model achieves F1 scores around 70% during cross validation, but performs badly on the test set.

All results were obtained using grid search through model hiperparameters (if the model has it) and k-folded cross-validation. For the Logistic Regression and linear SVM hiperparameters were normalization factors (C). For the Naive Bayes models hiperparameters were smoothing factors for hypothesis generation (α). For the Nearest Neighbours model hiperparameter was number of neighbours (n).

Classifier implementations from *scikit-learn* python module version 0.14 have been used to evaluate the models.

We have chosen two best classifiers: one discriminative (Logistic Regression) and the other generative (Multinomial Bayes). Then we used Mutual Information for feature selection on our word vectors. As mentioned earlier we used $n = 50, 100, 500, 1000, 5000$ most influential words of every class. We compared the results with the classifiers using all words. Results are shown in graphs 1 and 2.

Logistic Regression classifier is not working better with smaller number of features. Similar results had author in (McCallum and Nigam, 1998). Bayes classifier, on the other hand, had 1% better F1 score on 20 classes dataset with $n = 5000$ words with the highest mutual information.

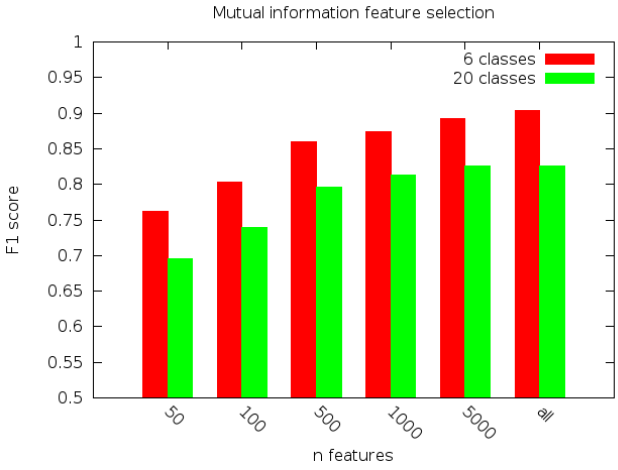


Figure 1: Comparison of Mutual Information feature selection with Logistic Regression

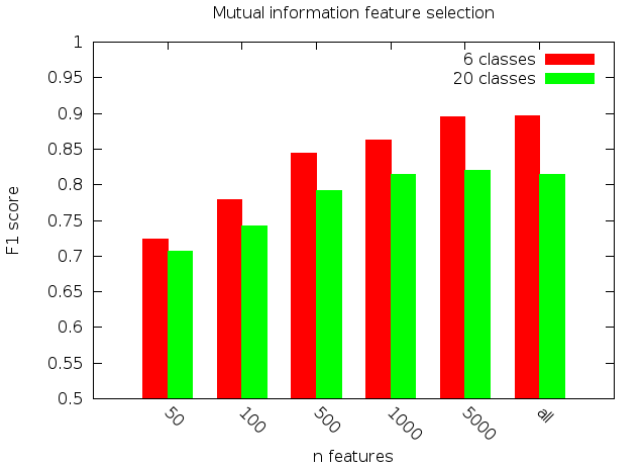


Figure 2: Comparison of Mutual Information feature selection with Naive Bayes

6. Conclusion

The best results for 6 classes dataset are 90.5% F1 score, obtained using Logistic Regression classifier with C=0.1. The same classifier was used to get the best result for 20 classed resulting in F1 score of 82.6%. This work has shown that in this particular case scores are not noticeably better when using Mutual Information feature selector, probably because tf-idf is used, instead of word counts. Results from this work are comparable with the ones presented in (McCallum and Nigam, 1998).

As seen in this work, text classification is one type of classification where bayes models and linear discriminative models work better that other sophisticated methods. Also feature dimension is much higher than the number of training examples, so the probability that the data is lineary separable is high.

Considering that some of the newsgroups in 20 Newsgroups dataset contain very similar words (e.g comp.sys.ibm.pc.hardware and comp.sys.mac.hardware), F1 score of over 80% is a pretty good result.

Further improvements could be achieved using different feature selection methods or changing the weighting scheme in document representation. Some possibilities worth considering can be found in (Forman, 2003) and (Soucy and Mineau, 2005).

References

George Forman. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*, 3:1289–1305, March.

Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features.

Shoushan Li, Rui Xia, Chengqing Zong, and Chu-Ren Huang. 2009. A framework of feature selection methods for text categorization. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 692–700, Stroudsburg, PA, USA. Association for Computational Linguistics.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

Andrew McCallum and Kamal Nigam. 1998. A comparison of event models for naive bayes text classification.

Pascal Soucy and Guy W. Mineau. 2005. Beyond tfidf weighting for text categorization in the vector space model. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 1130–1135, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Latent Semantic Indexing

Maja Buljan, Zolik Nemet

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{maja.buljan,zolik.nemet}@fer.hr

Abstract

Latent semantic indexing (LSI) is presented and evaluated as an information retrieval method. We give a brief overview of the underlying concepts and principles behind LSI, and describe the techniques involved in creating such a model, both in regards to building a word-document frequency matrix from a large text corpus, as well as the algebraic techniques employed in revealing high-order occurrence between sets of words, and comparing document representations in a high-dimensional vector space. We demonstrate the process by building a semantically-aware information retrieval system that implements LSI, which is then evaluated by its performance on the task of matching documents to queries from a given test set.

1. Introduction

Given a task of identifying and retrieving a set of documents relevant to a certain query, the simplest approach would be to model queries and documents as bag-of-words representations and perform comparisons to measure the degree of similarity between them. Such an approach, however, would fail to account for such linguistic features as polysemy and homonymy, which allow for different wordings of similar concepts, and vice-versa. Such an oversight leads to information loss and, in turn, decreased recall and precision, as user queries fail to retrieve relevant documents due to the system's inaccessibility of knowledge of the inherent expressiveness of natural language.

Latent Semantic Indexing (LSI) is an information retrieval method that extracts and models the latent semantic relations between sets of words implied by their contextual usage. By performing statistical computations on information extracted from large textual corpora, LSI discovers and implements information about high-order co- occurrence, i.e. previously unknown relations between words revealed by the similarity of their contexts, imbuing the system with real-world knowledge of a natural language's vocabulary, and thus refining and improving search results.

In this paper, we describe the construction of an LSI model for English language document search and retrieval. A brief overview of the field is given in Chapter 2. In Chapter 3, a more in-depth description of the model is given, including building a frequency matrix by processing the base text corpus, as well as the underlying algebraic techniques employed in the model's creation. The system is tested on the given set of documents and queries and the results of the evaluation are given in chapter 4.

2. Related work

2.1. Vector space models of semantics

Vector space semantic models were first introduced by G. Salton and his colleagues in the 1970s. The main idea behind the concept is expressed by the distributional hypothesis: words that occur in similar contexts tend to have similar meanings (Harris, 1954). Language models based on this idea aim to model the co-occurrence of words in contexts and documents by building a frequency matrix that maps documents and similar constructs to a high- dimensional vector space. The problem of matching a document to a query (represented in the same vector space as a pseudo-document) is then reduced to choosing an algebraic method for comparing two points – or their matching vectors – in the given space.

2.2. Latent semantic analysis

Latent semantic analysis (also called LSI in the context of information retrieval) was first presented by S. Deerwester et al. in 1990, with the purpose of further improving on the attempts to automatically build a model that would be able to more accurately reflect human knowledge.

By applying the algebraic technique of singular value decomposition (SVD) to a frequency matrix, LSI models the latent semantic relations between words and contexts, reduces the dimensionality of the document vectors by mapping the representations to a lower-dimension vector space, and acts as a smoothing technique, minimising the impact of noise in the data by retaining only those elements of the frequency matrix shown to have higher information content.

LSA has been extensively tested in various NLP tasks, such as synonym detection, semantic priming, document classification and retrieval, etc. (Landauer et al., 1997), and found to give good results in tasks requiring a level of discrimination between synonym, antonym, meronym, and other semantic relations between words.

3. Building the system

3.1. Processing the dataset

In order to implement a functioning LSI system, we first needed to build a term-document frequency matrix which would be used to model co-occurrence between words and discover latent relations between words and contexts. The matrix was built by analysing the WaCkypedia corpus, a pre-processed and lemmatised collection of English Wikipedia articles, sorting all unique vocabulary terms by descending frequency of occurrence, disregarding a set of common English stop words, and selecting the top n_t terms to use as the base for our term-document matrix.

We had originally envisioned to use the $n_t = 10\,000$ most frequently occurring words in the given corpus, but due to restrictions of a temporal nature, we settled on experimenting by using the first $n_t = 150$ and $n_t = 2\,250$ most frequently occurring words, which indubitably had an impact on the final evaluation results. The occurrence of the selected n_t words was then counted throughout the available documents ($n_d = 2\,700\,000$) to create a frequency matrix $\mathbf{F}$ ($n_t \times n_d = 2\,250 \times 2\,700\,000$, i.e. $150 \times 2\,700\,000$).

3.2. Singular value decomposition

Once the frequency matrix $\mathbf{F}$ was built, the next step was to perform singular value decomposition. SVD expresses $\mathbf{F}$ ($n_t \times n_d$, rank r) as:

$$\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}^T$$

where $\mathbf{U}$ ($n_t \times r$) and $\mathbf{V}^T$ ($r \times n_d$) are orthogonal matrices of left- and, respectfully, right-singular vectors of $\mathbf{F}$, such that $\mathbf{U}^T\mathbf{U} = \mathbf{V}^T\mathbf{V} = \mathbf{I}$, and Σ ($r \times r$) is the diagonal matrix of singular values.

$$\begin{bmatrix} \begin{bmatrix} \\ \mathbf{u}_1 \\ \\ \end{bmatrix} & \dots & \begin{bmatrix} \\ \mathbf{u}_r \\ \\ \end{bmatrix} \end{bmatrix} \cdot \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_r \end{bmatrix} \cdot \begin{bmatrix} [& \mathbf{v}_1 &] \\ & \vdots & \\ [& \mathbf{v}_1 &] \end{bmatrix}$$

Each column vector in $\mathbf{U}$ may be regarded as the representation of a latent concept, assigning weights to certain preferred words of the selected vocabulary. The column vectors of $\mathbf{V}^T$ contain the concept weights for each document. The singular values of Σ indicate the relevance of each latent concept in the reconstruction of $\mathbf{F}$. Each document may, therefore, be expressed as a weighted sum of latent concepts.

Given that the values in Σ signify the weights particular concepts (vectors of $\mathbf{U}$) are assigned in $\mathbf{F}$, by selecting the top k ($k < r$) values of Σ, and disregarding all other values, along with the matching vectors of $\mathbf{U}$ and $\mathbf{V}^T$, it is possible to reconstruct a reduced, rank k version of $\mathbf{F}$ that best approximates the original matrix, which may be expressed as:

$$\mathbf{F'} = \mathbf{U}_k\Sigma_k\mathbf{V}_k^T$$

As a result of this reduction, the once sparse matrix $\mathbf{F}$ is now replaced by its denser equivalent, $\mathbf{F'}$, in which certain words that earlier appeared in certain contexts may now be expressed to do so with greater or lesser frequency, and some that did not originally appear at all in the corpus-based frequency matrix may now have an above-zero value of occurrence frequency, leading to the assumption that SVD and reduction to k have revealed hitherto hidden co-occurrence relations between certain words and certain contexts. Therefore, not only does the vector space model illustrate context-based relations between words, but SVD reveals higher-order occurrence relations between words that were lacking in the corpus originally used for modelling word relations.

The resulting matrices $\mathbf{U}_k$ and Σ_k are then used as the basis of a vector space of reduced dimensionality, to which, when once built, the vectors of documents and queries for comparison are mapped.

3.3. Document and query representations

The given test collection comprised of a set of European Media Monitor articles, uniformly formatted but otherwise left in their raw textual form, and a set of phrases and sentences to be used as queries. In order to be able to use the "world knowledge" matrix described in previous subsections as aid in comparing queries to documents, both sets would first need to be modelled into a compatible format. Both documents and queries would be represented by frequency vectors of n_t elements, matching the n_t most frequent words selected from the vocabulary of the original dataset, and containing the occurrence counts of each word in the given document (or query).

Since the n_t reference words were properly lemmatised, while the documents of the test collection were not, in order to accurately match word occurrences to their lemmatised counterparts, a simple substring-matching algorithm was implemented.

Once we had built frequency vectors $\mathbf{d}$ ($1 \times n_t$) for all the given documents and queries, we mapped them to the reduced vector space of our LSI by multiplying them with the reduced left-singular vector matrix and the inverse of the reduced singular value matrix gleaned from the SVD and reduction process:

$$\mathbf{d'} = \mathbf{d}^T\,\mathbf{U}_k\,\Sigma^{-}1_k$$
$$\mathbf{q'} = \mathbf{q}^T\,\mathbf{U}_k\,\Sigma^{-}1_k$$

$$\mathbf{d'} = < d_1, d_2, \dots, d_k >$$
$$\mathbf{q'} = < q_1, q_2, \dots, q_k >$$

The given terms and documents could now be regarded as points in the vector space, and, using the chosen algebraic method of comparison, could be given similarity scores to be used in the final relevance judgement.

3.4. Comparing the vectors

In order to compare documents to queries, the cosine similarity was used, using the angle between two vector representations as a measure of similarity:

$$\begin{aligned} \cos(\mathbf{d'}, \mathbf{q'}) &= \frac{\sum_{n=1}^{k} d_i \cdot q_i}{\sqrt{\sum_{n=1}^{k} d_i^2 \sum_{n=1}^{k} q_i^2}} \\ &= \frac{\mathbf{d'} \cdot \mathbf{q'}}{\sqrt{\mathbf{d'} \cdot \mathbf{d'}}\sqrt{\mathbf{q'} \cdot \mathbf{q'}}} \\ &= \frac{\mathbf{d'}}{||\mathbf{d'}||} \cdot \frac{\mathbf{q'}}{||\mathbf{q'}||} \end{aligned}$$

For each pair, a real value between 0 and 1 was given, where 1 indicated a perfect match, while 0 indicated maximal disparity. For each query, documents were ranked by relevance according to this value, in descending order.

Table 1: System scores, compared as dependent on the choice of n_t (*n* most frequent terms) and *k* (*k* highest singular values)

n_t and *k*	R-Precision	Mean Average Precision
150; 10	0.098	0.031
150; 25	0.106	0.036

4. Evaluation

4.1. Results

The final performance of our system depended on several factors adjustable at various stages during the process of building the system. We assume the selection of a value for n_t for the top *n* most frequent words in the vocabulary was one of the more important factors, but due to the computationally intensive task of building the frequency matrix, we did not experiment further with this feature once after settling on an n_t of 150, i.e. 2 250 words.

Once the SVD of the frequency matrix was performed, we performed the reduction for several values of *k*. Mapping the query and document vectors to the reduced vector space was performed individually for each of these values, and the performance of the system with regards to standard evaluation metrics (R-Precision, Mean Average Precision) was calculated. The results are shown in Table 1.

4.2. Comments on the implementation

We believe the results of this system might be considerably improved by making a few minor adjustments during the building process.

Firstly, as mentioned earlier, the choice of *n* most frequent words might have a significant impact on the overall performance of the system, since it may reasonably be expected that a larger subset of the vocabulary will provide more options for informative word co-occurrence. Unfortunately, we were unable to test this assumption on values of n_t different from our eventual $n_t = 150$, i.e. 2 250 due to the amount of computation time our less-than-optimal implementation demanded. We would like to experiment further with n_t to find out what range of values would improve performance, or even be computationally feasible.

Additionally, a significant drawback of our system, and one we became aware of only long after the building of the frequency matrix **F** was well under way, was the fact that the lemmatised WaCkypedia corpus discriminated between lexically identical terms distinguishable only by a leading upper-case letter, and, unknowing of this feature of the corpus at the time of implementation, so did our word match-and-counter. The end result was that our selection of n_t most frequent terms included duplicates such as "lady" and "Lady", or "castle" and "Castle". Obviously, fixing this small error with massive consequences would undoubtedly lead to an improvement in our system's performance.

Finally, another somewhat disappointing feature of this implementation is the rudimentary substring matching algorithm used while counting occurrences of the top n_t terms in the test sets. Ideally, we would use a WordNet-based library to match all possible lexical variants of a term to its lemma, but in its current state, our system performs a basic substring check and comparison, which is bound to be too strict or too lenient in some cases.

5. Conclusion

Latent semantic indexing (LSI) is an efficient vector space model of semantics whose greatest advantage over similarly designed systems is the ability to identify and implement indirectly implied semantic relations between words and contexts, which simpler systems would most likely overlook. For this reason, LSI-based systems boast better IR performance than its word-to-word-matching counterparts.

We built and evaluated an LSI system, and the results showed that the system's overall performance is dependent on several user-adjustable features whose values were chosen largely arbitrarily.

As part of future work, we intend to experiment with these features, and try to devise rational heuristics according to which their values ought to be chosen. We also plan to make some crucial adjustments to our system, in order to rectify some of the drawbacks mentioned earlier in the paper, and measure their influence on the system's performance.

References

Deerwester et al. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*.

Landauer, Foltz, and Laham. 1998. An introduction to latent semantic analysis. *Discourse Processes*.

Manning, Raghavan, and Schutze. 2007. An introduction to information retrieval. *Cambridge University Press*.

Turney and Pantel. 2007. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*.

Tweet Classification

Jan Bzik, Matija Stepanić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{jan.bzik, matija.stepanic}@fer.hr

Abstract

This paper explains the method implemented in a project as a part of a course "Text Analysis and Retrieval". Goal of this project is to classify tweets in Croatian into five classes (News, Sport news, Music news, Private messages, Other) using machine learning algorithms. Tweet dataset is acquired using Twitter API. Also, subset of Croatian twitter corpus is used. Tweets are manually labeled by two annotators, features are extracted and passed to SVM, KNN and Naive Bayes classifiers which are trained and tested. Python and its libraries are used for the implementation.

1. Introduction

Twitter is a microblogging social network that allows people to send and receive messages, post multimedia content and follow each other. Twitter is referred as a microblogging social network because it has a text length restriction. Maximum number of characters in one tweet is 140. This fact makes text mining and text analysis on Twitter a tiring effort. Twitter as a social network in 21st century is a website with heavy traffic. Tweets are posted every second and Twitter users are bombed with data. Tweet classification is a process that could make usage of Twitter more user friendly.

Classification is a process of labeling instances from a finite set of labels. Machine learning algorithms allow this to be done automatically. In grammar, inflection is the modification of a word to express different grammatical categories such as tense, mood, voice, aspect, person, number, gender and case. Croatian is a highly inflective language so this is also a problem that needs to be addressed within solving this task. There are several tweet classification methods that use meta-information about tweets from other sources like WordNet or "What the Trend" website. Such services for Croatian are not available or still in a development process.

Method described in this paper uses a Bag of Words model for feature extraction. In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. Information about popularity of a particular tweet (retweet count and favorite count) and information about structure of a tweet (presence of particular characters) is also considered.

2. Similar work

2.1. Twitter Trending Topic Classification

Lee et al.[1] use 2 approaches for tweet classification. First approach is already mentioned Bag of Words model. Second approach is network-based classification. In text-based classification tf-idf weights are used to classify tweets using a Naive Bayes Multinomial classified. In network-based classification, top 5 similar topics are identified based on the number of common influential users. In this paper, "What the Trend" website is mined and popular *trending* topics are identified and tweets mentioning these topics are retrieved.

2.2. Short text classification in Twitter to improve Information Filtering

Sriram et al.[2] select a feature set which follows the definitions of classes. They classify tweets into five classes: News, Events, Deals, Private messages and Opinions. They extract percentage and currency characters which point that a tweet is a deal. Also they extract @ character which is used on Twitter as a reference to a user so its presence is a strong indicator of a tweet being a private message. Time event phrases, opinionated words and slang are also extracted and passed as features to Naive Bayes classifier.

3. Method

3.1. Data collection and labeling

Croatian Twitter corpus is searched for users connected with topics such as sports, news and music. These users are typically Twitter profiles of big newspaper magazines and internet sports or music portals. Paralelly, using Twitter API, new tweets posted by these users are also retrieved. Using these process labeling is made automatically by storing tweets, e.g. from user dnevnikhr, to a file named news.txt. Tweets in which character @ is at the beggining of text are labeled as private messages. Other tweets are labeled as other.

3.2. Data preprocessing and modeling

Tweets contain a lot of noise produced by using slang and bad grammar so they need to be filtered and normalized. This step is called preprocessing data. First, every letter is made lowercase and punctuation marks are filtered. After that, using regular expressions, URL-s are replaced with empty strings. Tokenization is the next step. Tokenization is the process of breaking up a sentence or a stream of text into words, symbols etc. These elements are called tokens. Stopwords are defined and removed from token list before stemming. Stopwords list is defined according to the nature of Croatian language itself. Tokens are passed to a porter stemmer using Croatian dictionary for stemming. Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form. Stemming

is needed for normalization of words so that they could be counted.

Described preprocessing process is done for every tweet in dataset. After preprocessing word coefficient that describes importance of a word by class is calculated for every class. First, words are counted in each tweet belonging to a certain class. Then coefficient is calculated:

$$c(w) = \log \frac{s}{t}$$

where s is the number of words in all tweets belonging to a certain class and t is the number of occurences of a word *w* in all tweets belonging to a certain class. This coefficient shows the importance of a word within a class.

3.3. Feature extraction

Created dataset contains 500 tweets belonging to class sport news, 800 tweets belonging to class news, 400 tweets belonging to class private messages, 700 tweets belonging to class music news and 600 tweets belonging to class other. Twenty percent of dataset is excluded and saved for classifier testing. Eighty percent of dataset comprises training set and it is used for extracting the features and learning the classfier.

Favorite count and retweet count are first features that form feature vector. These features are taken because big Twitter accounts like newspaper or sports portal account tend to have more favorited and retweeted tweets. Presence of a character @ is binary, one if it is present and zero if it is not. Other features are calculated according to words that occur in the tweet. For every word it is checked if it exists in the dictionary of words connected to a certain class. If a word exists its contribution is calculated like: $contribution(w) = \frac{1}{c}$. Contributions are sumed up and make another element inside a feature vector. Contributions are calculated for classes: sport news, news and music news.

3.4. SVM classification

SVM (Support Vector Machine) is a supervised learning model that is used heavily in pattern recognition, computer vision and data mining. It is used for classification and regression analysis. Support Vector Machines in 2D find an optimal hyperplane that separates two classes of points with the maximum margin.

Features extracted in the previous step are passed to a SVM and optimization process is started. SVM learning process is stopped when the convergence threshold is satisfied.

Table 1: Results: SVM

CLASS	PRECISION	RECALL	F1-SCORE
SPORT NEWS	0.6	0.54	0.57
NEWS	0.68	0.63	0.65
PRIVATE MESSAGES	0.88	0.62	0.73
OTHER	0.56	0.72	0.63
MUSIC NEWS	0.61	0.66	0.63
TOTAL	0.65	0.64	0.64
ACCURACY		0.64	

3.5. Naive Bayes classification

In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution. Here, Gaussian Bayes classification is used

Table 2: Results: GNB

CLASS	PRECISION	RECALL	F1-SCORE
SPORT NEWS	0.39	0.64	0.49
NEWS	0.47	0.04	0.07
PRIVATE MESSAGES	0.69	0.85	0.76
OTHER	0,40	0.91	0.55
MUSIC NEWS	0.66	0.31	0.42
TOTAL	0.50	0.46	0.38
ACCURACY		0.46	

3.6. K - Nearest Neighbours classification

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: Euclidean, Manhattan etc. Standard Euclidean distance is the most common choice.

Table 3: Results: KNN

CLASS	PRECISION	RECALL	F1-SCORE
SPORT NEWS	0.36	0.39	0.38
NEWS	0.57	0.66	0.61
PRIVATE MESSAGES	0.67	0.3	0.41
OTHER	0,48	0.51	0.49
MUSIC NEWS	0.57	0.51	0.54
TOTAL	0.53	0.52	0.51
ACCURACY		0.52	

4. Testing

4.1. Results and statistical measures

In information retrieval and classification problems certain statistical measures are calculated and present the successefulness of an algorithm.

Precision is defined with:

$$\frac{true_{positive}}{true_{positive} + false_{positive}}$$

Recall is defined with:

$$\frac{true_{positive}}{true_{positive} + false_{negative}}$$

Accuracy is defined with:

$$\frac{true_{positive} + true_{negative}}{total}$$

F1 score is defined with:

$$\frac{2 * true_{positive}}{2 * true_{positive} + false_{positive} + false_{negative}}$$

We use a weighted F1 score. It is obtdained by calculating metrics for each label, and finding their average, weighted by support (the number of true instances for each label).

After training SVM classifier, test set is used to predict labels of instances passed to trained classifier. Results for SVM classification are shown in Table 1 and for other two classifiers trained in Table 2 and Table 3.

Results are not satisfying due to noisy data and to simple and primitive features used for training the classifier. Results could be improved by new features, e.g. POS-tagging. Also, results could be improved by using a Croatian version of WordNet or by using a big dictionary of words semantically connected to definition of classes.

5. Graphical web interface

As a part of this project web GUI (Figure 1.) was developed using Python based Django framework for back - end, Javascript based AngularJS framework for front - end and Twitter API for real time tweet acquisition.

User inputs a word preceded by "#" or "@" and by this defining a hashtag or a usernam. If the input is a hashtag the number of tweets, containing that hashtag , defined in a "Integer" text box is taken from Twitter. If the input is a username tweets posted by that user are taken from Twitter and analyzed.

After tweets are pulled, they are analyzed and it is possible to see the results presented in a graph for each of the three classifiers trained in this project.

Currently, web application is filled with "dummy" data but our goal is to make it operable in the future. It is available at `http://tvitaj.ga`.

6. Conclusion

Tweet classification is a task of short text classification and such tasks are sensitive to noise. Features used for classification often follow from class definitions. Language is also an important factor. Many libraries, labeled datasets, web dictionaries etc. are available for languages like English and French but are not available (or special authentification is required) for Croatian. POS-tagging, lemmatization, stemming and similar preprocessing steps are hard without mentioned tools and reduce the quality of feature extraction. However, classification is possible for certain tweets that use words that have high frequency within a certain class.

Between three classifiers used in this project, SVM gave the best results. Further improvements of a system for automatic classification of tweets would be building a bigger and better dictionary of words connected to a certain class and extracting features that have semantical background connected to definition of a class.

References

Ramanathan Narayanan, Md. Mostofa Ali Patwary, Ankit Agrawal, and Alok Choudhary 2001. *Twitter Trending Topic Classification*. Department of Electrical Engineering and Computer Science, Northwestern University

Bharath Sriram, Dave Fuhry, Engin Demir, Hakan Ferhatosmanoglu, Murat Demirbas 2010. *Short Text Classification in Twitter to Improve Information Filtering* Proceedings of the 33rd International SIGIR conference on research and development in information retrieval

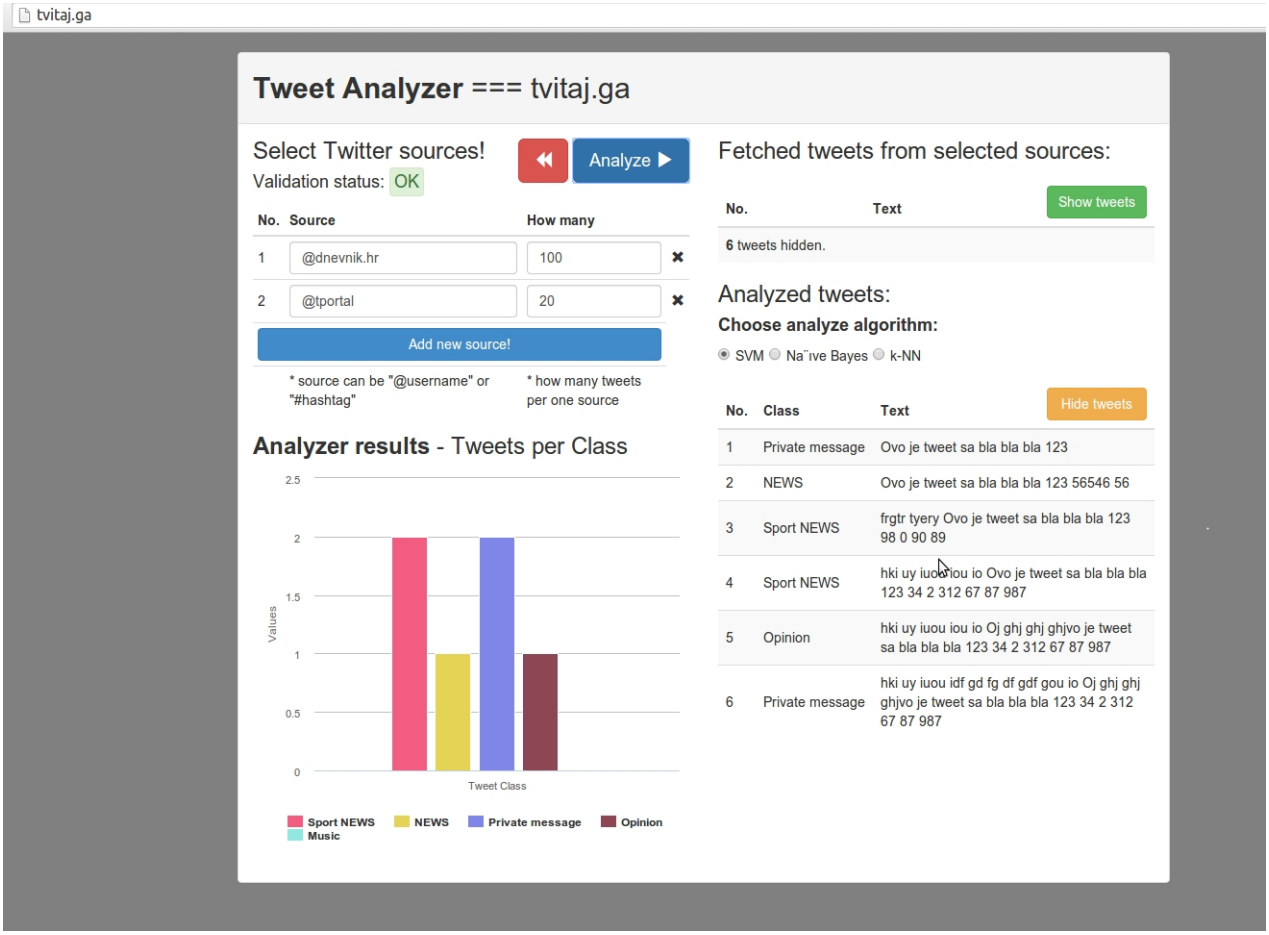


Figure 1: Web GUI

Link analysis algorithms (HITS and PageRank) in the information retrieval context

Damir Ciganović-Janković, Teon Banek, Dario Miličić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{damir.ciganovic-jankovic, teon.banek}@fer.hr, dario.milicic@gmail.com

Abstract

The research is done within the Text analysis and retrieval course as a project assignment. We implemented an information retrieval (IR) system in Python using a binary vector-space retrieval model and evaluated it along with both PageRank and HITS algorithms on the set of queries with labeled relevant documents. We used European Media Monitor (EMM) test collection as the evaluation dataset. For link analysis algorithms to work, we constructed a bidirectional link between documents that were determined to be similar using cosine similarity function. Performance was measured using standard IR Metrics, R-Precision and Mean Average Precision. After evaluation of all three algorithms we draw a conclusion that link analysis algorithms do not improve information retrieval models for this data set.

1. Introduction

Surfing through the Internet has never been easier. Only a few clicks and several keyboard keys pressed and we are already getting what we want from our favourite web browser. We could easily say that browsers these days work so good that it seems like they know what we know even before we type it. To realize how it is possible to produce such good results, we must look into algorithms being used for retrieving relevant web pages for a given query. These algorithms are called link analysis algorithms. The aim of this project is to build an information retrieval system that ranks the documents based on the analysis of between-document links using PageRank and HITS link analysis algorithms (Manning et al., 2008a). Primary hypothesis is that with the help of PageRank and HITS algorithms, system should get better results in retrieving relevant documents for a given query than using just the binary vector space model. Our goal is to test that thesis and to check which one of the two algorithms will score better results under the same conditions. This research will help us better understand how web searching is done.

2. Dataset description

For testing and evaluation we used European Media Monitor (EMM) IR test collection (Newsbrief, 2014). It contains documents, queries and relevance judgements for each of the queries. Every document contains text that is a news report from some news report company. Documents are stored as .xml files. Queries are represented as .txt file containing one or more sentences. Relevance judgements for all queries are given in a single .txt file. Every line in the file is a triplet separated by spaces. Triplets consist of: query name, article name and relevance label. There is in total 1741 documents and 50 queries, but there is only 48 queries with their relevance judgements attached.

3. Information retrieval metrics

For evaluation of ranked retrieval results there are several different metrics. In this paper, primary metrics were R-precision and Mean Average Precision (Manning et al., 2008b).

R-precision metric is calculated as follows. For given set of known relevant documents *Rel* for query q we produce ranked results by one of the algorithms. Then we take first *R* values *bestR* from produced results where *R* is the number of relevant documents for query q. For every document in *bestR* we check if it is one of the relevant documents and if it is, we add it to the *r*. R-precision value is r/R. R-precision calculates precision for one point only so to get the whole model evaluation, we calculated arithmetic mean of R-precision values for every query.

Mean Average Precision is an arithmetic mean of average precisions for every query. Average precision is calculated as follows. If the set of relevant documents for a query $q \in Q$ is $\{d_1, d_2, \ldots, d_m\}$ and R_k is the set of ranked retrieval results from the top result until you get to the document d_k, then average precision *AP* is:

$$AP(q) = \frac{1}{m} \sum_{k=1}^{m} Precision(R_k) \tag{1}$$

4. Binary vector space model

Building information retrieval system requires a few steps, including some natural language processing methods. For a given text, we extracted all words by splitting text on every white-space and striping all punctuation marks. After that, the words are stemmed using the Porter Stemmer to narrow down the difference between words with same root (Porter, 1980). Before the next step we must take into account that in the text articles, there is always a set of words that is common for every text regardless of topic discussed. That set is called stop-words set and it is also removed from the set of article words. We compiled that set of words by combining two sets (Ranks, 2014; Lextek, 2014). Total number of stop-words is 571.

What we have by now is all different words from a document. After we did that for every document, we compiled a set of all words from all articles. Now, to build a binary vector from the article, we search for every word from all-words set and if it is found in an article, we set the value

Table 1: *R-precision* and *Mean Average Precision* values for given threshold, binary vector model

Sim. threshold	R-precision	MAP
0.05	0.2247	0.2397
0.1	0.2247	0.2397
0.15	0.2179	0.2094
0.2	0.1484	0.1306
0.25	0.0567	0.0540

for that word to 1, and 0 otherwise (Manning et al., 2008c). This is what makes it a binary model. Total vector length is therefore the same as the number of words in all-words set, which is 21057.

5. Determining whether a document is relevant for a given query

To determine if a document is relevant for a given query we first converted the query text into a vector in the same way as we converted documents into vectors. Query vector size is the same as the document vector size because we use the same all-words set. After that, we determine for every document if it is relevant for a given query or not, and then we sort all relevant documents in specific order based on the algorithm used.

For each algorithm, we have measured performance using mentioned IR metrics. All calculations were done in Python 2.7 on a Mac OS X, version 10.9.3 using 1.3 Ghz Intel Core i5.

5.1. Binary vector space model without link analysis algorithms

For binary vector space model alone, we used cosine similarity between query vector and every document vector. If the cosine similarity is above a certain threshold, we label that document as relevant for the query, and not relevant otherwise. After that step, we rank our relevant documents based on cosine similarity value, from highest to lowest to get ranked results.

We set the similarity threshold to 0.1. It should be noted that similarity and relevance threshold are the same value. To determine that value of the threshold, we evaluated the system with thresholds in range of 0.05 to 0.29 (with 0.01 step), and chose the most appropriate one. The reason why we chose 0.1 for the threshold value is because the R-precision and MAP measures are significantly dropping when threshold value is getting higher from 0.1, but not significantly rising when threshold is getting lower. Results can be seen in Table 1.

5.2. Binary vector space model with PageRank

PageRank is one of the most widely known link analysis algorithms. It was developed by Larry Page and Sergey Brin as part of a research project about a new kind of search engine (Brin and Page, 1998). It is also used commercially at Google Inc. In short, PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

For PageRank and HITS algorithms to work, we needed to determine links between documents in our data set. Link analysis algorithms work on a graph consisting of documents as vertices and edges as links between those documents. To convert our data set to such a graph, we constructed links based on document similarity. Similarity score was calculated for each document pair using cosine similarity with documents represented as binary word vectors. If the similarity score was above a certain threshold then we considered those documents linked with a bidirectional link. Resulting graph was represented in memory as an adjacency matrix. In the adjacency matrix *M*, if the documents *i* and *j* are linked then *M[i][j]* and *M[j][i]* are equal to 1, otherwise it is equal to 0. This adjacency matrix is used for both PageRank and HITS algorithms.

PageRank is an algorithm that doesn't depend on the query as input. It is run during preprocessing stage, before a query is given to the system for evaluation. After the algorithm has finished, every document will have a PageRank score that will determine its ranking during query evaluation. The higher the score, the more important the document is.

When a query is given, it is first converted to a vector as described in Section 4. This query vector is then compared to all documents from highest to lowest PageRank score. If the query vector and the document vector are similar, according to cosine similarity and the relevancy threshold, then the document is declared relevant. This way the more important documents are higher in the result list. The key difference between this approach and the approach without link analysis algorithms is that the similarity score has no impact on the ranking of documents. Similarity score is simply needed to determine if the document is relevant or not.

This makes a certain scenario possible that may seem as an error in the information retrieval system. The reasoning behind this approach should explain why such behaviour is wanted. Suppose that a similarity score of a query vector and a document is very high, for example similarity score $\alpha >= 0.99$. We might believe that this document should be at the top of the result list but with PageRank this doesn't have to be the case. This is actually a very common scenario when searching the web with Google. Consider searching for *"read it"* on Google's search engine. The result will actually be the social networking website *reddit* even though there is a page that is more similar to the given query. The reason for this is that *reddit* has a much higher PageRank score which implies that the user most probably wanted *reddit* as a search result in the first place. In other words, PageRank score determines implicit probability that the user wants a certain page as a search result.

When calculating MAP and R-precision, better performance was achieved by the basic vector space model than the model which includes the PageRank algorithm. This implies that our hypothesis, which states that link analysis algorithms will improve results, may not be correct. Results can be seen in Table 2.

Table 2: *R-precision* and *Mean Average Precision* values for given similarity threshold, binary vector model and PageRank.

Sim. threshold	R-precision	MAP
0.05	0.0064	0.0174
0.1	0.0444	0.0666
0.15	0.1612	0.1682
0.2	0.1270	0.1093
0.25	0.0463	0.0369

Table 3: *R-precision* and *Mean Average Precision* values for given similarity threshold, binary vector model and HITS.

Sim. threshold	R-precision	MAP
0.05	0.0060	0.0111
0.1	0.0260	0.0406
0.15	0.1196	0.1048
0.2	0.1302	0.1156
0.25	0.0567	0.0412

5.3. Binary vector space model with HITS

Hyperlink-Induced Topic Search (HITS) is a link analysis algorithm developed by Jon Kleinberg (Kleinberg, 1999). Unlike PageRank, in HITS there are two scores that have to be evaluated for each document. These scores are called hub score and authority score. A page has a high hub score if it points to pages with high authority scores. Also, a page is a good authority if it is pointed by pages with high hub scores. These scores are interdependent: Hub score of a page equals the sum of the authority scores of the pages it links to; Authority score of a page equals the sum of hub scores that point to it.

In our data set, each link is bidirectional which means hub and authority scores are not going to differ. Different pages will have different scores because of different number of links but their hub and authority values will be equal given that their starting values are equal. Another difference from PageRank is that HITS takes a query as input. This will make HITS much slower than PageRank as it calculates the scores only after a query is given. However, HITS doesn't calculate hubs and authorities on the entire graph. Rather, it takes a subgraph that will be called a base set. The base set depends on the query input. Base set is generated as follows: When a query is given it is compared for similarity against all documents in the data set. All documents that have a similarity score above the threshold are added to a root set. In addition to the documents in the root set, the base set contains all the documents that are linked to from the root set. When these documents are added, the base set is ready for evaluation.

Evaluation of the binary vector model with HITS shows that it doesn't improve search results by any metric. Also, it evaluated the slowest as the entire algorithm had to be performed for each query. Results can be seen in Table 3.

6. Conclusion

It is easy to see by all metrics that both link analysis algorithms have failed to improve search results for the given data set. Another metric that wasn't measured is time but the difference was noticeable with PageRank and binary vector model being almost equally fast and HITS being the slowest.

Link analysis algorithms weren't designed for these kind of data sets but it is important to be aware that it is sometimes hard to predict without trying where certain tools can be helpful. In this research opportunity, however, our original hypothesis has not been proven.

References

Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. Technical report, Stanford University.

Jon Kleinberg. 1999. Authoritative sources in a hyperlinked environment. Technical report, Cornell University.

Lextek. 2014. Lextek stop word list.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Shutze, 2008a. *Introduction to Information Retrieval*, chapter 21. Cambridge University Press.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Shutze, 2008b. *Introduction to Information Retrieval*, chapter 8. Cambridge University Press.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Shutze, 2008c. *Introduction to Information Retrieval*, chapter 14. Cambridge University Press.

EMM Newsbrief. 2014. Emm-ir-collection.

Martin Porter. 1980. Porter stemmer.

Ranks. 2014. Mysql stop word list.

Predictive Typing System

Matej Filković, Dino Koprivnjak, Petar Kovačević

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{matej.filkovic,dino.koprivnjak,petar.kovacevic}@fer.hr

Abstract

We survey some of the most widely-used algorithms for smoothing models for language n-gram modeling. We present a comparison of two smoothing techniques. We investigate how training data size affect the relative performance of these methods, which is measured through the cross-entropy of test data.

1. Introduction

A *language model* is a probability distribution *p(s)* over string *s* that describes how often the string *s* occurs as a sentence in some domain of interest [1].

The central goal of the most commonly used language models, trigrams models, is to determine the probability of a word given the previous two words: $p(w_i|w_{i-2}w_{i-1})$. The simplest way to approximate this probability is to compute:

$$p_{ML}(w_i|w_{i-2}w_{i-1}) = \frac{c(w_{i-2}w_{i-1}w_i)}{c(w_{i-2}w_{i-1})} \quad (1)$$

i.e the number of times the word sequence $c(w_{i-2}w_{i-1}w_i)$ occurs in some corpus of training data divided by the number of times the word sequence $c(w_{i-2}w_{i-1})$ occurs [1]. This value is called *the maximum likelihood* estimate.

The maximum likelihood estimate can lead to poor performance in word prediction application. *N-grams* only work well for word prediction if the test corpus looks like the training corpus and in real life, it often doesn't [2]. To address this problem smoothing is used.

Smoothing is technique for adjusting the maximum likelihood estimate of probabilities to produce more accurate probabilities, by adjusting the low probabilities such as zero probabilities, but they also attempt to improve the accuracy of the model as a whole[1]. In this work, we carry out two smoothing techniques: Additive smoothing and Kneser-Ney smoothing.

This paper is structured as follows: We give brief introduction to n-gram models in Section 2. In Section 3, we discuss the performance metrics with which we evaluate language models. In Section 4, we describe Additive and Kneser-Ney smoothing. In Section 5, we present the results of all of our experiments. In Section 6 we summarize the most important conclusions of this work. Finally, we have developed a text editor with the predictive typing capabilities.

2. N-gram language model definition

The most widely-used language models, are n-gram language models[1]. For a sentence *s* composed of the words $w_1...w_l$, we can express $p(s)$ as

$$\begin{aligned} p(s) = p(w_1| < s >) \times p(w_2| < s > w_1) \\ \times ... \times p(w_l| < s > w_1...w_{l-1}) \\ \times p(< /s > | < s > w_1...w_l) \end{aligned} \quad (2)$$

where the token $< s >$ is a distinguished token representing the beginning of the sentence, and $< /s >$ signals the end of the sentence. In an n-gram model, we make the approximation that the probability of a word depends only on the identity of the immediately preceding n - 1 words, giving us[1]:

$$p(s) = \prod_{i=1}^{l+1} p(w_i|w_1....w_{i-1}) \approx \prod_{i=1}^{l+1} p(w_i|w_{i-n+1}^{i-1}) \quad (3)$$

where w_i^j denotes the words $w_i...w_j$ and where we take w_{-n+2} through w_0 to be $< s >$. To estimate $p(w_i|w_{i-n+1}^{i-1})$ we count how often token w_i follows the context of w_{i-n+1}^{i-1} and divide it by the total number of times the history occurs, i.e to take[1]:

$$p_{ML}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^i)}{c(w_{i-n+1}^{i-1})} = \frac{c(w_{i-n+1}^i)}{\sum_{w_i} c(w_{i-n+1}^i)} \quad (4)$$

where $c(w_i^j)$ denotes how often the *n*-gram w_i^j occurs in some training data. This estimate is the *maximum likelihood* estimate of the probabilities on a training set, and smoothing algorithms typically produce probabilities near to this estimate.

3. Cross-entropy and perplexity

The most common metric for evaluating a language model is the probability that the model assigns to test data, or the derivative measures of *cross-entropy* and *perplexity*[1]. For a test set *T* composed of the sentences $(t_1, ..., t_{l_T})$ we can calculate probability of the test set *p(T)* as the product of the probabilities of all sentences in the set: $p(T) = \prod_{k=1}^{l_T} p(t_k)$. The cross-entropy $H_p(T)$ of a model $p(t_k)$ on data *T* is defined as $H_p(T) = -\frac{1}{W_T} log_2 p(T)$ where W_T is the length of the text T measured in words[1].

The perplexity $PP_p(T)$ of a model *p* is the reciprocal of the average probability assigned by the model to each

word in the test set T, and is related to cross-entropy by the following equation:

$$PP_p(T) = 2^{H_p(T)} \tag{5}$$

Lower cross-entropies and perplexities denote better language model.

4. Smoothing techniques for language modeling

4.1. Additive smoothing

One of the simplest types of smoothing used in practice is additive smoothing. To avoid zero probabilities, we pretend that each n-gram occurs slightly more often than it actually does: we add a factor δ to every count, where typically $0 < \delta \leq 1$. Thus, we set[1]:

$$p_{add}(w_i|w_{i-n+1}^{i-1}) = \frac{\delta + c(w_{i-n+1}^{i})}{\delta\|V\| + \sum_{w_i} c(w_{i-n+1}^{i})} \tag{6}$$

where V is vocabulary, or set of all words considered.

4.2. Kneser-Ney smoothing

Kneser-Ney smoothing method is an extension of absolute discounting where the lower-order distribution that one combines with a higher-order distribution is built in a novel manner. To compute probability *p(w)* using Kneser-Ney smoothing method, we have to compute the continuation probability $p_{continuation}(w)$ that tells us how likely is word w to appear as a novel continuation. To estimate $p_{continuation}$ we count how many time does w appear as novel continuation normalized by the total number of word bigram types, i.e to take:

$$p_{continuation}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|} \tag{7}$$

Using the $p_{continuation}(w)$ we have the following Kneser-Ney smoothing algorithm[2] for bigrams:

$$p_{kn}(w_i|w_{i-1}) = \frac{max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) p_{continuation}(w_i) \tag{8}$$

where λ is a normalizing constant. We estimate λ as follows

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})}|\{w : c(w_{i-1}, w) > 0\}| \tag{9}$$

where d is absolute discounting, and typically $0 \leq d \leq 1$. For higher order n-gram we use following recursive formulation[2]:

$$p_{kn}(w_i|w_{i-n+1}^{i-1}) = \frac{max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1}) p_{kn}(w_i|w_{i-n+2}^{i-1}) \tag{10}$$

where

$$c_{KN}(\bullet) = \begin{cases} count(\bullet) & \text{for the highest order} \\ continuationcount(\bullet) & \text{for lower order} \end{cases} \tag{11}$$

Continuation count is number of unique single word contexts for $\bullet$.

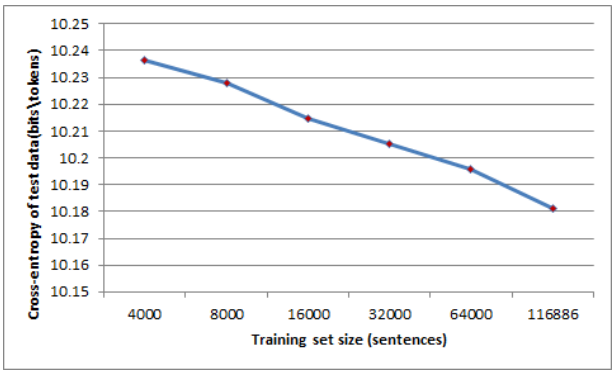


Figure 1: Cross-entropy of the Kneser-Ney algorithm on the test set over various training set sizes for bigram language model

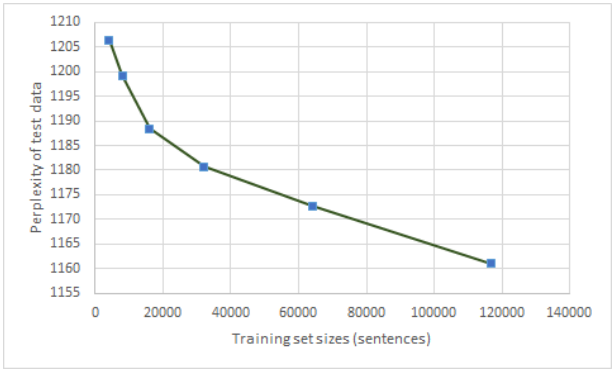


Figure 2: Perplexity of the Kneser-Ney algorithm on the test set over various training set sizes for bigram language model

5. Results

As mentioned earlier, we evaluate smoothing methods through their cross-entropy and perplexity on test data. For our training data we used first four books of Game of Thrones pentalogy, while fifth book was used as test set. Figure 1, shows the cross-entropy while figure 2 perplexity of the Kneser-Ney smoothing algorithm , over a variety of training set sizes for bigram language model. We see that cross-entropy and perplexity decreases steadily as the training set used grows in size. Figure 3, shows the cross-entropy while figure 5 perplexity of the additive smoothing algorithm, over variety of training set sizes for bigram language model. We see that cross-entropy and perplexity decreases steadily as the training set grows in size. Comparing results for both algorithms, it is possible to conclude that the model, which uses Kneser-Ney smoothing algorithm has lower cross-entropy and perplexity which means that this language model predicts better on unseen test data.

6. Conclusion

Smoothing is a fundamental technique for statistical modeling, important not only for language modeling. We analysed performance of two different smoothing algorithms through the cross-entropy and perplexity of test data. From

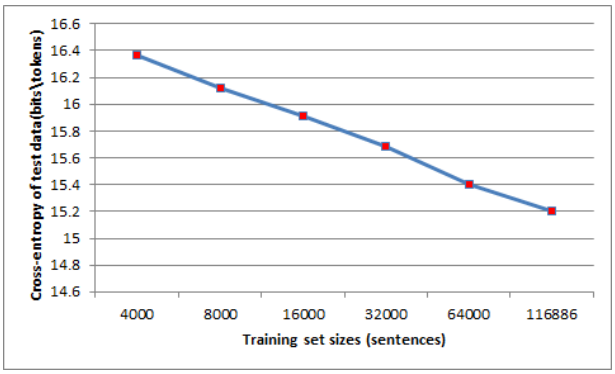


Figure 3: Cross-entropy of the additive smoothing algorithm on the test set over various training set sizes for bigram language model

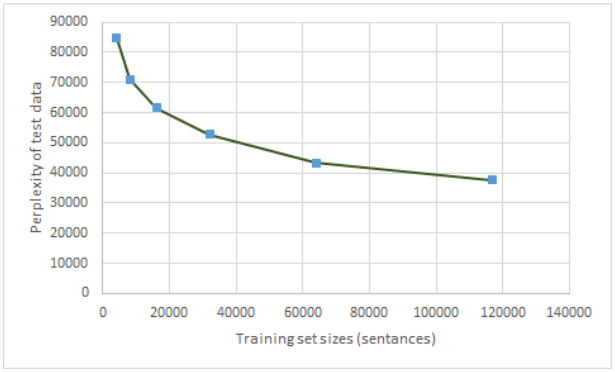


Figure 4: Perplexity of the additive smoothing algorithm on the test set over various training set sizes for bigram language model

the presented results it is possible to conclude that Kneser-Ney smoothing algorithm preforms better than additive smoothing algorithm. It is also visible that larger amounts of training data produce better results. Testing on various test sets has shown that the model performs better the closer the test set is to the training set. A good language model should perform wel onl varying test data.

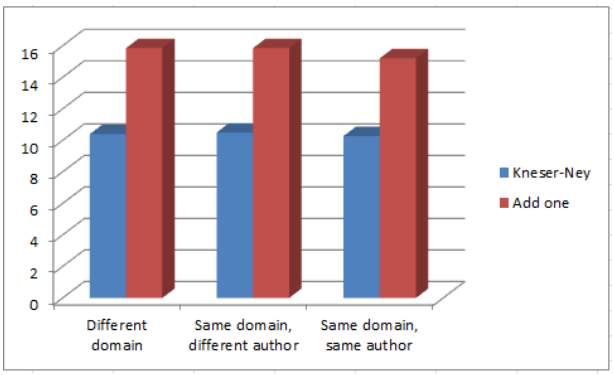


Figure 5: Cross-entropy of the additive smoothing algorithm over different test sets.

References

Stanley F. Chen, Joshua Goodman 1999. *An empirical study of smoothing techniques for language modeling. Computer Speech and Language*. Carnegie Mellon University, Pittsburgh

Dan Jurafsky and Christopher Manning 2014. *Natural Language Processing*, online Coursera course, Stanford University, 2014 - https://www.coursera.org/course/nlp

Gobinda G.Chowdhury *Natural Language Processing*, in Annual Review of Information Science and Technology, 37. pp. 51-89

Daniël de Kok 2011. *N-grams" in Natural Language Processing for the Working Programmer*, chapter 3 , 2011 - http://nlpwp.org/book/index.xhtml

Text Classification

Zvonimir Ilić, Ivan Hakštok

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
zvonimir.ilic@fer.hr, ivan.hakstok@fer.hr

Abstract

This document provides description of main features of automated text classification system that we have built. This system works as a binary classifier and classifies documents from 20-newsgroup dataset. Although we will emphasize machine learning techniques that are implemented, this document will also provide explanation of technical capabilities and implementation of our automated classifier. Reader will also be provided with performance statistics of this classifier system and technical limitations of system.

1. Introduction

Our automated classifier system works on 20-newsgroup dataset. This classifier system is binary classifier where one newsgroup's articles are classified against another one. User can choose between two classifying techniques: logistic regression and näive Bayes. User can also choose two different feature selection techniques: mutual information feature selection and χ^2 feature selection.

Evaluation of classifier is also very important so user can compare result of one classifying technique against another one with different parameters.

2. Classifying algorithms

In this section we will describe machine learning algorithms used for classification.

2.1. Logistic regression

Logistic regression is classifying technique that provides probabilistic output. In our classifier system it works with tf-idf numerical features. First step to apply logistic regression is to generate tf-idf vector for each document. This is achieved with following expressions:

$$TF(t,d) = \frac{1}{2} + \frac{\frac{1}{2} \cdot f(t,d)}{\max\{f(w,d) : w \in d\}},$$

$$IDF(t,D) = \log \frac{|D|}{|d \in D : w \in d|},$$

where t denotes term, d denotes document, $f(t,d)$ denotes frequency of term t in document d and D is set of all documents. After tf-idf transformation each document is represented with n-dimensional vector, where n is size of vocabulary and value for each term is given with $TF(t,d) \cdot IDF(t,D)$.

Goal of optimizing logistic regression parameters is to minimize cross entropy error which is defined as

$$E = -\sum_i \{y_i \ln h(\mathbf{x_i}) + (1-y_i)(1 - \ln h(\mathbf{x_i}))\},$$

where $h(\mathbf{x})$ is defined as

$$h(\mathbf{x}) = \frac{1}{1 + \exp\{-(\mathbf{w}^T \cdot \mathbf{x} + w_0)\}},$$

and $\mathbf{w}, w_0$ are parameters that are being optimized.

Parameters of logistic regression are optimized with gradient descent. Gradient descent is repeated until difference between two consecutive iterations of cross entropy error is less then 10^{-3}.

To prevent overfitting, it is useful to examine different values of regularization parameter λ. With lambda we increase loss function so model has better generalization properties. Error function in that case becomes

$$E = -\sum_i \{y_i \ln h(\mathbf{x_i}) + (1-y_i)(1-\ln h(\mathbf{x_i}))\} + \frac{\lambda}{2} \cdot \mathbf{w}^T \cdot \mathbf{w}.$$

2.2. Näive Bayes

Näive Bayes is probabilistic model classification which uses Bayes theorem for classification. In our automated classifier we use following expression:

$$h_{map} = \operatorname{argmax} P(c) \cdot \prod_{1 \le k \le n_d} P(t_k|c).$$

Essentially, we are finding class that gives *maximum aposteriori* probability for some document. *A priori* probability can be estimated as

$$P_C = \frac{N_C}{N},$$

where N_C is number of documents belonging to class C, and N is number of total documents. Conditional probability $P(t_k \mid c)$ is estimated as relative frequency of term t in documents belonging to class c :

$$P(t_k|c) = \frac{T_{ct}}{\sum T_{ct'}}$$

where T_{ct} is the number of occurrences of t in training documents from class C. However, previous equation is not suitable for proper classifier evaluation. It is reasonably to assume that some term t, which occurs in testing set, will not occur in class C in training set, so we have to modify previous equation. To eliminate zeros we use Laplace smoothing, and conditional probability becomes

$$P(t_k|c) = \frac{T_{ct} + 1}{\sum T_{ct'} + |V|}$$

where V is set of words.

Terms	Mutual information
space	0.2101
nasa	0.1148
orbit	0.1034
launch	0.0899
earth	0.0877
moon	0.0746
circuit	0.0701
flight	0.0691
shuttle	0.061

Table 1: Terms with highest mutual information in classification sci.space against sci.electronics.

3. Feature selection

In this section will be examined feature selection techniques that are used to reduce dimensionality.

3.1. Mutual information

Mutual information measures how much information the presence or absence of some term provides about class of a document. In other words, we want to evaluatehow much information the presence/absence of a term contributes to making the correct classification of a document. It is obvious that we can use only terms with high mutual information, and discard rest of them because they provide small amount of information and cause sparsity in machine learning model.

Mutual information for some random variables U and C is

$$I(U,C) = \frac{N_{11}}{N}\log_2\frac{N_{11}N}{N_{1.}N_{.1}} + \frac{N_{01}}{N}\log_2\frac{N_{01}N}{N_{0.}N_{.1}} +$$

$$\frac{N_{10}}{N}\log_2\frac{N_{10}N}{N_{1.}N_{.0}} + \frac{N_{00}}{N}\log_2\frac{N_{00}N}{N_{0.}N_{.0}}.$$

where N is number of documents, N_{tc} is number of documents that have distinct class label c and distinct indicator of existence of term t. In our classifier, the terms with highest mutual information in classification sci.space against sci.electronics are in Table 1.

3.2. χ^2 feature selection

χ^2 test is used in statistics to determine whether two events are independent. We say tha events A and B are independent if $P(A \cdot B) = P(A) \cdot P(B)$. However, in feature selection χ^2 test is used to measure independence of occcurence of class and occurence of term. This measure can be calculated with the following expression:

$$\chi^2(D,t,c) = \sum_{e_t\in 0,1}\sum_{e_c\in 0,1}\frac{(N_{e_te_c} - E_{e_te_c})^2}{E_{e_te_c}}$$

where N is observed frequency and D and E are expected frequency. This measure is used to sort terms and get the most important ones in a binary classification.

3.3. Feature selection and impact on performance

We noticed that feature selection improves time performance of classification while accuracy, precision and recall of classifier do not significantly change. As we can see on these pie diagrams, results of classification on tst set with Näive Bayes are almost the same with significantly better performance. When we apply feature selection on logistic regression we can notice almost the same effect: performance is highly improved with precission and recall almost equaly with entire set of features. Feature selection is much more important for logistic regression because in logistic regression we have to calculate dot product of vectors with very high dimensions(≈ 15000).

4. Classifier evaluation

Classifier is evaluated by evaluating accuracy, precission, recall and F-score. Training and testing sets are separate sets and thes do not overlap. Size of training set is 60% of all examples and size of testing set is 40% of all examples. Precision, recall and F1-score for pair of newsgroups and classification algorithms can be seen in Table 2. Huge impact on measures like accuracy, precision, recall and F-score has selection of newsgroups. In Table 2. we can notice that first pair of newsgroups (comp.sys.mac.hardware / sci.electronics) has better recall and F1-score than second one (comp.windows.x / comp.os.ms.windows.misc). This is beacause articles in newsgroups comp.windows.x and comp.os.mswindows.misc have more similarities (both newsgroups are about operating system Windows and "windows" is word with highest mutual information) then articles in newsgroups comp.sys.mac.hardware and sci.electronics. This pattern can be observed in entire dataset: similar newsgroups classifications have higher number of false negatives or false positives and therefore precision, recall and F-score are lower.

5. Implementation

In this section we will explain implementation of automated classifier system.

5.1. Dataset pre-processing

In classification we used pre-processed representation of 20-newsgroups dataset. This pre-processed interpretation is done by authors of dataset. Entire dataset is represented with sequence of tuples (d, w, c) where d is index of document, w is index of word in vocabulary and c is number of occurences of word w in document d. Each newsgroup has its own index and separate file has sequence of these indices. Position of newsgroup index is index of file and number(index) is index of newsgroup.

5.2. Technology

Our automated classifier system is implemented using Visual Studio 2012 IDE in programming language C# with Git as collaboration and source control technology. System also provides simple but intuitive GUI which is made using Windows Forms API. Main idea behind implementation was to create scalable system which can easily be upgraded with additional machine learning algorithms and feature selection techniques.

Algorithm	Newsgroups	Accuracy	Precision	Recall	F1-score
Logistic regression ($\lambda = 5$)	comp.sys.mac.hardware/sci.electronics	0.905	0.885	0.927	0.906
Logistic regression ($\lambda = 2$)	comp.windows.x-comp.os.ms/windows.misc	0.891	0.894	0.887	0.891
Näive Bayes	comp.sys.mac.hardware/sci.electronics	0.893	0.849	0.953	0.898

Table 2: Results of testing classification algorithms on test set. In these examples feature selection hasn't been used.

Algorithm	Newsgroups	Accuracy	Precision	Recall	F1-score
Logistic regression ($\lambda = 5$)	sci.electronics/sci.space	0.927	0.918	0.926	0.922
Näive Bayes	comp.sys.mac.hardware/sci.electronics	0.887	0.844	0.945	0.892

Table 3: Results of testing classification algorithms on test set. In these examples is used mutual information with 100 most significant features.

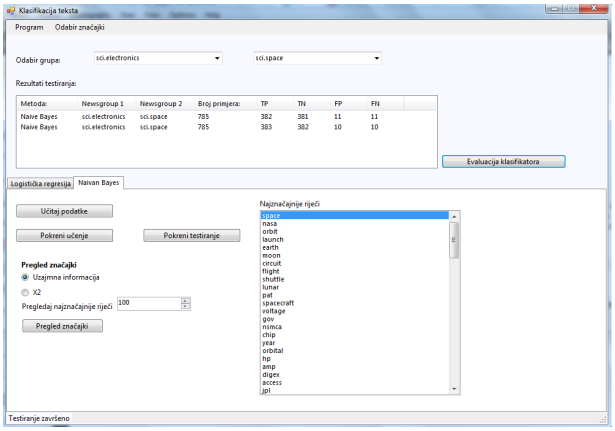


Figure 1: Main Window

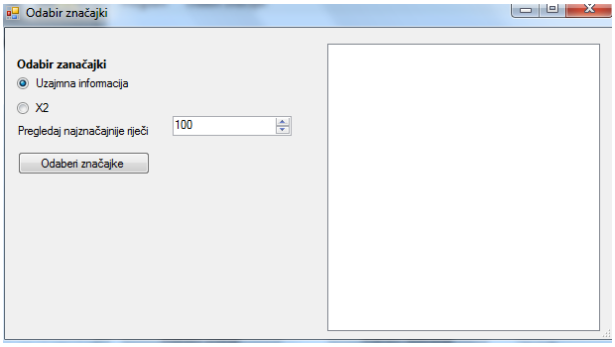


Figure 2: Feature selection window

5.3. GUI and user instructions

On 1 we can see main window and its tabs for logistic regression and NB. This user interface also contains section with test results of various classifications. There user can compare different machine learning algorithms and same algorithms with different hyperparameters and regularizations.

In 2 we can see window for feature selection. User can choose which method of feature selection wants to use and number of feuters with which classifier is going to be fed.

6. Conclusion

In this paper we see how machine learning algorithms like logistic regression and Näive Bayes can be used for automated text classification. Mentioned algorithms also have much better performance when they are combined with feature selection technique, because these techniques reduce time needed for model training. Implemented automated classifier can be upgraded with more classification algorithm and their result can be compared against other classification algorithms or even the same algorithm with different hyperparameters or regularization factor. This use case can already be engaged by user in logistic regression (user can compare tests of logistic regression with different regularization parameters).

Acknowledgements

We want to thank community on stats.stackexchange.com for many useful tips in implementation of machine learning algorithms and feature selection.

References

Chris Manning, Heinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press. Cambridge

Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer

Sentiment Analysis

Dino Jović, Kristijan Ivančić, Lana Lisjak

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{dino.jovic,kristijan.ivancic,lana.lisjak}@fer.hr

Abstract

This document proposes a method and shows results for sentiment analysis of user reviews on various products. The projects consist of two parts. The first part shows classification of new reviews as positive or negative, and the second demonstrates prediction of user ratings based on textual reviews. Additionally, encountered problems in failed models are shown and explained, with suggested corrections.

1. Introduction

The goal of this project is to build a sentiment classification and rating prediction for textual user reviews on certain products. Reviews were taken from multiple domains, including books, DVDs, electronic devices etc. The classification model was evaluated in terms of accuracy, precision, recall and F1-score. The rating prediction model was evaluated using a regression metric called Pearson correlation.

2. Preparing data

Initial data sets consisted of textual reviews, which were parsed and tokenized by splitting the string on all white lines. This created a list of tokens. The next step consisted of removing all non-letter characters. Unfortunately, there was not enough time to implement a search function for emoticons. The list of strings was further reduced by removing all stop words. Creating a dictionary of this list amounted to 44000 words. This dictionary contained words with equal or similar meaning, like book-books etc. This problem was solved by stemming the dictionary, which decreased the number of words to around 30 000. Data was prepared by merging all domains in positive and negative sets. Both sets were sorted by random numbers to mix domains. Those random numbers were always seeded with 1234567890, for better comparison during testing. The first 75% of sets were taken for training and the remaining 25% were taken for testing.

3. Classification

A pivot model was chosen for classification. Its prediction is based on the number of positive and negative words in text. Pivots were chosen in the following way:

- Computing $TF * IDF$ for every word in every document.
- Creating dictionary of words.
- For every word computing

$$tfidfpoz = \sum_{review==positiv} TF * IDF \quad (1)$$

.

- For every word computing

$$tfidfneg = \sum_{review==negativ} TF * IDF \quad (2)$$

.

- Sorting dictionary by

$$tfidfpoz - tfidfneg \quad (3)$$

.

First words in a dictionary were only used in negative reviews, and last words were only used in positive reviews. Words used in both classes will have sorting value near zero. Words not often used will also have sorting value near zero. The parameter $lambda$ was used to describe which percentage of best and worst words were used on the testing set. At first it was decided that a review was positive if sentiment equals:

$$sentiment = \sum_{inGood} 1 - \sum_{inBad} 1 \quad (4)$$

And then if:

$$sentiment > review.wordCount * lambda \quad (5)$$

An equivalent procedure was taken for determining if a review was negative. If both conditions were false, the review was neutral. During later stages of testing it was difficult to test properly without neutral reviews, so the model was simply changed to count if there were more good or bad words. That model achieved an F1-score of 0.74. In later stages there was an insufficient memory issue, so all words with less than 5 appearances in the learning set were removed from the dictionary. The final dictionary consisted of 7011 words. The model was optimized by an F1-score over the parameter lambda ranging form 1% to 50% with a step of 1%. Best results were achieved with lambda set at 39% with the F1-score of 0.736.

4. Regression

For predicting user ratings a logistic regression model was used. Possible ratings were on a scale from 1 to 5, but the dataset had no reviews with a rating of 3, so the training and evaluation was done without them, using only labels 1, 2, 4 and 5. The ratings themselves were not uniformly distributed. The distribution was around 2:1:1:2, which was considered acceptable.

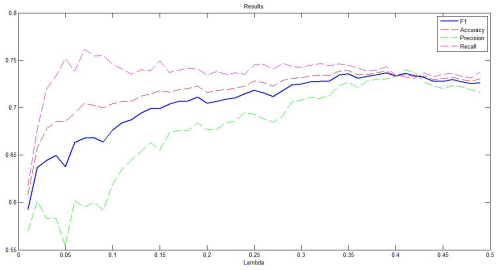


Figure 1: Graph showing accuracy, precision, recall and F1-score against parameter lambda (percentage of words used).

Table 1: Example of bad and good words (positive and negative words)

Domains	Pearson's coefficient r
books	0.31
DVDs	0.30
electronic devices	0.19
kitchen	0.26

Table 2: Example of bad and good words (positive and negative words)

Good words	Bad words
enjoy	book
also	kagan
size	return
food	bad
cook	wast
famili	movi
price	machin
vs	unit
pan	bore
best	poor
perfect	disappoint
grill	worst
workout	erin
music	product
great	just
excel	replac
use	softwar
love	month
film	warranti
easi	card

Linear regression was used for ranking reviews on scale 1 to 5. The created models predicted ratings for unlabelled reviews, showing good results. When reviews with high probability of being correctly classified were moved to the training set, the training set became increasingly non-uniform, with distributions close to 2:1:1:10. With this situation in mind, semi supervised learning was not used.. Another problem was memory allocation, since the unlabelled set was much bigger than the labelled set. Domains were split in four groups : books, DVDs, electronic devices and kitchen.

5. Results

Figure 1 shows the results of classification. A slow rise of F1-score is evident until $lambda$ reaches 0.39, which is considered the best score this model has reached. Table 1 shows the results of regression as a Pearson's coefficient. Pearson's coefficient is a measure of dependence between two quantities. If Poison's coefficient is positive, correlation between real and predicted label exists. Smaller absolute value of Poison's coefficient implicates stronger correlation. Result would be better if real number were used for rating reviews, not just integers from 1 to 5. Domains for electronic devices and kitchen appliances are wider and have more ambiguity (silent headphones and silent processor), so results for these domains are worse when compare to other domains.

6. Conclusion

The proposed model showed acceptable results, beating benchmarks such as majority class classifier and random class classifier. Of course, this model couldn't beat state of the art classifiers. Regression also showed acceptable results considering the scope of this project. Also, correcting the non-uniform distribution of input data and adding user reviews with missing scores would further improve the results of the proposed model.

References

Kory Becker. 2013. Tf*idf in c# .net for machine learning - term frequency inverse document frequency.

John Blitzer, Mark Dredze, and Fernando Pereira. 2007. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL*, volume 7, pages 440–447.

Inc. GitHub. 2014. Sharpkit.learn - machine learning library for .net (c#, f#).

ssotnyk. 2012. Snowball stemmers on c#.

Like It or Not? Classification and Grading of Sentiment in User Reviews

Azzaro Mujić, Dinko Osrečki

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{azzaro.mujic,dinko.osrecki}@fer.hr

Abstract

We consider the problem of automatic classification of documents by their sentiment. Sentiment analysis aims to determine the attitude of the writer with respect to some topic or the overall contextual polarity of a document. Focusing on on-line reviews from four different domains (books, DVDs, electronics and kitchen and housewares) we present simple probabilistic supervised learning algorithm for classifying those reviews as *positive* or *negative*. The classification of reviews is predicted by the Naive Bayes classifier. Output of the algorithm is not only the label, but also the score in range of 1 to 5 stars. The algorithm achieves an average accuracy of 85% when evaluated using 10-fold cross-validation. The accuracy ranges from 76% for books reviews to 92% for DVDs reviews. We also present results in terms of precision, recall and F1 score for the classification model and mean absolute error, mean squared error and Pearson's coefficient for the rating prediction model.

1. Introduction

Documents can be categorized in various ways. Sentiment classification is a special case of text categorization, where the criterion of classification is the attitude expressed in the text, rather than the topic. Useful applications of sentiment analysis are in question answering, business intelligence applications, recommender systems and recognizing "flames" (abusive newsgroup messages) (Spertus, 1997). Another application is text summarization. Imagine yourself at home on a rainy, gloomy night looking for a movie to cheer you up. You have heard of a movie called *American Pie*, so you might go to Google and enter the query "American Pie". In this case Google retrieves about 70 million results. It would be great to know which fraction of these results recommend *American Pie* as a great movie to watch. With an algorithm for automatically classifying a review as "like" and "don't like" it would be possible for a search engine to give you that information (Pang et al., 2002; Turney, 2002).

There are at least two approaches for classifying sentiment in documents. The first approach is to count positive and negative terms in a document. Document is considered positive if there are more positive than negative terms, and negative if it contains more negative terms. One can find positive and negative terms in the *General Inquirer*[1] - a dictionary that contains information about English word senses including tags that label them as positive or negative (Kennedy and Inkpen, 2006).

The second approach is to use machine learning to determine the sentiment of the reviews. This approach is carried out in this work. We trained the Naive Bayes classifiers for each domain and used unigrams (single words) as features. We also included some specific bigrams. To be more precise, we included only bigrams that contain a combination of negation with another feature word (e.g. *not_good*).

Table 1: Number of documents in each category.

Category	Number of documents
Books	2000
DVDs	36741
Electronics	15153
Kitchen and appliances	18785

2. Dataset

We used dataset presented by (Blitzer et al., 2007) in their work on sentiment analysis. They constructed a dataset by selecting Amazon product reviews for four different product types: books, DVDs, electronics and kitchen appliances. Reviews are provided in form of *XML* files. Each review consists of a review title, review text, rating, reviewer name, reviewer location, product name, product type, unique identification label and Amazon standard identification number. Reviews with rating greater than three were labeled positive and those with rating less than three were labeled negative. We had 72679 documents available for training and validation. Table 1 shows the number of documents per each category.

[1] http://www.wjh.harvard.edu/~inquirer/homecat.htm

Table 2: Results in terms of accuracy, precision, recall and F1 score for the classification model and mean absolute error, mean squared error and Pearson's coefficient for the rating prediction model for each domain.

Measure/Category	Books	DVDs	Electronics	Kitchen and appliances
Accuracy	0.7625	0.9164	0.8427	0.8732
Precision	0.8359	0.9506	0.9501	0.9581
Recall	0.6528	0.9523	0.8333	0.8774
F1 score	0.7331	0.9515	0.8879	0.9159
MAE	1.0790	0.5483	0.7860	0.6298
MSE	2.7611	1.0404	1.8092	1.4578
Pearson's coef.	0.5832	0.6712	0.6771	0.7140

3. Preprocessing

Here, it is the main task to prepare the dataset for classification process. We made a script that takes *XML* files described in section 2. and outputs transformed *JSON* files.

Script performs following processing steps:

1. ignores all the content between tags that are irrelevant for classification process (e.g. ⟨*unique_id*⟩, ⟨*asin*⟩, ⟨*product_name*⟩, ...);
2. processes the content between ⟨*review*⟩ ⟨*/review*⟩ tags in the following way:
   - maps emoticons to unique identifiers (e.g. :D to ⟨*happy*⟩, :(to ⟨*sad*⟩),
   - transforms each word to lowercase,
   - expands contractions (e.g. don't to do not),
   - transforms numbers into special identifier (e.g. 5 to ⟨*num*⟩),
   - performs tokenization, i.e. splits the text into bag of words,
   - removes from the text each word that is less than three characters long,
   - connects each negation with the neighbouring word on the right (e.g. not bad to not_bad).

At the end we got *JSON* file that contains following attributes:

- category of the review (e.g. books, DVDs),
- review rating on scale from 1 to 5,
- bag of words extracted from the review text after performing preprocessing steps described in this section.

4. Experiment and Results

For classification, we used Naive Bayes classifiers trained on the data from each domain. Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong independence assumptions between the features. We chiefly used unigrams (single words), but also bigrams of negated words as features. Data were preprocessed as described in section 3.

Table 2 shows results in terms of accuracy, precision, recall and F1 score for the classification model and mean absolute error, mean squared error and Pearson's coefficient for the rating prediction model for each domain.

The accuracy is the proportion of true results, and is given as:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}, \quad (1)$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives and FN is the number of false negatives.

Precision is defined as the number of true positives against all the positive results and is given as:

$$Precision = \frac{TP}{TP + FP}. \quad (2)$$

Recall is defined as the number of correctly classified instances against all the positive instances in the dataset and is given as:

$$Recall = \frac{TP}{TP + FN}. \quad (3)$$

From these values we have calculated the F1 score:

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}. \quad (4)$$

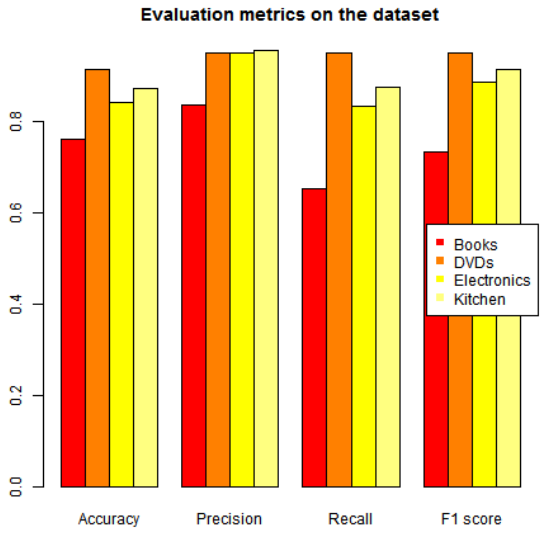


Figure 1: Evaluation metrics on each category of the dataset.

Mean absolute error (MAE) is given as:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y^{(i)} - \hat{y}^{(i)}|, \tag{5}$$

where $y^{(i)}$ is the true rating and $\hat{y}^{(i)}$ is the predicted value. Similarly, mean squared error (MSE) was calculated in the following way:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2. \tag{6}$$

Finally, we have calculated Pearson's correlation coefficient with the following formula:

$$r = \frac{\sum_{i=1}^{N}(x^{(i)} - \bar{x})(y^{(i)} - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x^{(i)} - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y^{(i)} - \bar{y})^2}}. \tag{7}$$

Figure 1 provides the reader with the graphical results of the experiment.

5. Demo

We wanted to make some interaction with the user, so we made a program that allows the user to write a review that will be processed. First, program loads the classifier (which is already trained as described in the section 4.) from file. After that, it reads in the input data, pre-processes it (as described in the section 3.) and transforms the document into features. Finally it outputs prediction whether the review is positive or negative and also gives the rating prediction - score between 1 and 5.

For example, if we use classifier trained on books dataset and the user provides the program with the following input: "This book is so boring. It really sucks.", program will output that the review is negative with score 1.319186.

6. Related work

With the abundance of blogs and social networks, opinion mining and sentiment analysis became a field of interest for many researches. A very broad overview of the existing work was presented in (Pang and Lee, 2008). In their survey, the authors describe existing techniques and approaches for an opinion-oriented information retrieval. However, not many researches in opinion mining considered blogs and even much less addressed microblogging.

In (Yang et al., 2007), the authors use web-blogs to construct a corpora for sentiment analysis and use emotion icons assigned to blog posts as indicators of users' mood. The authors applied SVM and CRF learners to classify sentiments at the sentence level and then investigated several strategies to determine the overall sentiment of the document. As the result, the winning strategy is defined by considering the sentiment of the last sentence of the document as the sentiment at the document level.

Read (2005) used emoticons such as ":-)" and ":- (" to form a training set for the sentiment classification. For this purpose, the author collected texts containing emoticons from Usenet newsgroups. The dataset was divided into "positive" (texts with happy emoticons) and "negative" (texts with sad or angry emoticons) samples. Emoticons trained classifiers: SVM and Naive Bayes, were able to obtain up to 70% of an accuracy on the test set.

In (Go et al., 2009), authors used Twitter to collect training data and then to perform a sentiment search. The approach is similar to (Read, 2005). The authors construct corpora by using emoticons to obtain "positive" and "negative" samples, and then use various classifiers. The best result was obtained by the Naive Bayes classifier with a mutual information measure for feature selection. The authors were able to obtain up to 81% of accuracy on their test set. However, the method showed a bad performance with three classes ("negative", "positive" and "neutral").

7. Conclusion

Sentiment analysis is relatively new in the context of research. Nevertheless, it is an important current research area. In this paper we showed that it is greatly

beneficial. According to section 4. we can see that if the dataset is well prepared then sentiment analysis performs well. Metrics like accuracy, precision, recall and F1 score are pretty high which means that our model can make good predictions on new reviews.

As it can be seen in the section 5., we provided demo program so that the user can enter review and get prediction whether it is positive or negative, as well as get predicted rating in the range between one to five. In our research our model works best when training and test data are drawn from the same distribution. However, when training and test data are drawn from different domains, metrics such as accuracy are decreased. In our future work, we plan to make sentiment analysis so that the training set can be drawn from the population different than the one from which the testing dataset is drawn. It will use structural correspondence learning algorithm that will automatically induce correspondences among features from different domains.

References

J. Blitzer, M. Dredze, and F. Pereira. 2007. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 440–447, Prague, Czech Republic, June. Association for Computational Linguistics.

Alec Go, Lei Huang, and Richa Bhayani. 2009. Twitter sentiment analysis.

Alistair Kennedy and Diana Inkpen. 2006. Sentiment classification of movie reviews using contextual valence shifters. *Computational Intelligence*, 22:2006.

Bo Pang and Lillian Lee. 2008. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1-2):1–135, January.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up?: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 79–86, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jonathon Read. 2005. Using emoticons to reduce dependency in machine learning techniques for sentiment classification. In *Proceedings of the ACL Student Research Workshop*, ACLstudent '05, pages 43–48, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ellen Spertus. 1997. Smokey: Automatic recognition of hostile messages. In *In Proc. IAAI*, pages 1058–1065.

Peter D. Turney. 2002. Thumbs up or thumbs down?: Semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 417–424, Stroudsburg, PA, USA. Association for Computational Linguistics.

Changhua Yang, Kevin Hsin-Yih Lin, and Hsin-Hsi Chen. 2007. Emotion classification using web blog corpora. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, WI '07, pages 275–278, Washington, DC, USA. IEEE Computer Society.

Klasifikacija tweetova

Pruša David, Plavčić Barbara, Crnjac Drago

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{david.prusa,barbara.plavcic,drago.crnjac}@fer.hr

Sažetak

Mikroblogging usluge poput Twittera su postale jednim od najpopularnijih načina za dijeljenje vlastitog mišljenja sa svijetom. Velika količina tweetova može preplaviti korisnika te otežati pronalazak zanimljivog i relevantnog sadržaja. Kako bismo se suočili s ovim problemom, stvorili smo sustav koji klasificira tweetove u jednu od tri kategorije (vijest, posao, privatna poruka) koristeći naivni Bayesov klasifikator.

1. Uvod

Usluga Twitter jest mikroblog usluga koja omogućava korisnicima da lakše dijele svoje mišljenje u obliku mikroblog poruka. Naziv mikroblog je nastao iz razloga što Twitter omogućava korisnicima dijeljenje poruka veličine do 140 znakova. Moguće je pratiti više korisnika od kojih želimo pratiti najnovije objave. Upravo iz ovog razloga, postoji mogućnost da korisnika preplavi ogromna količina tweetova te mu onemogućava pronalaženje relevantnih tweetova u toj "šumi" podataka. Cilj našeg rada jest omogućiti korisniku klasificiranje tweetova u jednu od tri kategorije (vijest, posao, privatna poruka) kako bi što lakše i brže došao do željenih informacija.

Postojeći radovi na ovu temu koriste meta-informacije iz izvora kao što su Wikipedia i WordNet [(0), 0)]. Mi predlažemo pristup u kojem najčešće korištene riječi iz pojedinih kategorija označavaju pripadnost tweeta toj kategoriji. Za svaki tweet izračunavamo koeficijent pripadnosti odredjenoj kategoriji. Budući da se većina tweetova iz dobivenog skupa podataka može smatrati privatnim porukama, najjednostavnije rješenje prilikom kojeg bismo imali točnost klasifikatora preko 99% jest to da svaki tweet klasificiramo kao privatnu poruku. Ovakvo rješenje ne bi imalo smisla osim na zadanom skupu podataka, te smo se odlučili koristiti naivni Bayesov klasifikator. Testovima smo potvrdili da klasifikator dobro generalizira te da smo izbjegli prenaučenost klasifikatora.

2. Priprema skupa podataka i odabir značajki

Dobiveni skup podataka se sastojao od datoteke veličine veće od 5 gigabajte u kojoj se nalazilo otprilike 17 500 000 tweetova. Svi tweetovi su bili napisani od strane 26 431 autora koji su najmanje imali po 1 tweet, a najviše 2840 tweetova. Kao što je već navedeno, najveći broj tweetova su bile privatne poruke u kojima se korisnici Twittera obraćaju jedni drugima koristeći znak "@" nakon kojeg slijedi korisničko ime osobe kojoj je poruka namijenjena.

2.1. Pročišćavanje i uredjivanje skupa podataka

U početku smo trebali očistiti xml od čudnih ascii znakova koji su nam narušavali performanse klasifikatora. XML je sadržavao neke simbole koji nisu služili ničemu prilikom klasificiranja, a parser ih nije mogao pročitati. Zato smo te simbole i znakove izbacili.

Za parsiranje je bio potreban korijenski element koji nije postojao. Svi tweetovi su bili zapisani u formatu jedan nakon drugoga. U početku smo koristili DOM parser, no ubrzo smo odustali od njega budući da je datoteka od 5.4 gigabajta bila prevelika za njega. Nakon dodavanja korijenskog elementa, odlučili smo se na korištenje SAX parsera kojemu je potreban korijenski element.

2.2. Odabir značajki

Odabir dobrih značajki jedan je od najbitnijih koraka prilikom izgradnje klasifikatora. Kao značajke smo odlučili koristiti: broj korisnika koji je navedeni tweet označio kao omiljeni (eng. favorite), broj korisnika koji je proslijedio navedeni tweet (eng. retweet), jezik na kojemu je tweet napisan, duljina tweeta, sadrži li navedeni tweet referencu na postojećeg korisnika Twittera (znak "@" popraćen korisničkim imenom osobe), koeficijent pripadnosti tweeta kategoriji vijesti, koeficijent pripadnosti tweeta kategoriji poslova.

Broj korisnika koji je navedeni tweet označio kao omiljeni jest dobra značajka budući da tweetovi koji pripadaju kategoriji vijesti ili poslovnih ponuda često imaju veću popularnost zbog velikog broja pratitelja (eng. followers) koji prate medijsku kuću ili tvrtku koja dijeli neku vijest ili obavijest o novim ponudama na tržištu. Iz istog razloga smo i broj retweetova koristili kao značajku. Često korisnici, kojima je materinji jezik hrvatski, koriste neki drugi jezik prilikom citiranja pjesme, filma i slično. Upravo iz ovog razloga, velika količina tweetova koji koriste strani jezik pripadaju kategoriji privatnih poruka u kojima obični korisnici dijele neki stih ili citat iz filma. Manja duljina tweetova takodjer u većini slučajeva označava pripadnost tweeta kategoriji privatnih poruka. Korisnici često dijele trenutnu radnju koristeći samo riječ ili dvije poput "jedem...", "sretan sam!" i slično. Tweetovi u kojima se referencira na nekog drugog korisnika Twittera pripadaju gotovo uvijek u kategoriju privatnih poruka iz razloga što se velike tvrtke najčešće ne bave koncentriranim dijeljenjem poruka s odredjenim korisnikom, već ciljaju šire mase korisnika svojim sadržajem. Koeficijenti pripadnosti tweeta kategoriji vijesti i poslovne ponude su jedne od najbitnijih značajki. Budući da tek malen broj tweetova iz zadanog

skupa od 17 500 000 tweetova pripada kategorijama vijesti ili poslovnih ponuda, morali smo smisliti način pomoću kojeg bismo pronašli što više primjera vijesti ili poslovnih ponuda. Ručno pretraživanje nije dolazilo u obzir budući da je broj tweetova prevelik. Pronašli smo nekoliko vijesti i poslovnih ponuda, nakon kojeg smo pretpostavili da korisnici Twittera, koji su jednom napisali vijest ili poslovnu ponudu, većinu vremena pišu takve tweetove. Pronašli smo 8 autora koji većinom pišu vijesti i 8 autora koji većinu vremena pišu o poslovnim ponudama, te smo od njih prikupili 8000 tweetova vijesti i 8000 tweetova poslovnih ponuda. Nakon toga smo izbacili riječi koje se pojavljuju u obje skupine tweetova poput: "a", "i", "u", "ali" i slično. Preostale riječi smo sortirali po broju ponavljanja u kategoriji te smo dobili težinu riječi koja govori koliko "jako" riječ pripada nekoj od kategorija. Potom smo za svaki tweet provjerili koje se riječi koriste te koliko se često te riječi ponavljaju u tweetovima klasificiranim u jednu od kategorija. Primjerice, ukoliko smo odredili da se riječ "politika" pojavila u tweetu, te da se ista ta riječ pojavila u skupu tweetova vijesti 125 puta, tada bismo koeficijentu pripadnosti tweeta vijesti dodali 125, isto tako ukoliko bismo u istom tweetu naišli na ciječ "cijena", tada bismo pogledali koliko se puta riječ "cijena" pojavila u skupu tweetova koji pripadaju kategoriji poslovne ponude, te dodali tu brojku u koeficijent pripadnosti tweeta poslovnoj ponudi.

3. Klasifikator

Koristili smo naivni Bayesov klasifikator koji je jedan od najpoznatijih klasifikatora općenito. Ovaj generativni model klasificira primjere koristeći Bayesovo pravilo (1) koje nam za svaki primjer daje vjerojatnost da taj primjer pripada odredjenoj klasi.

$$P(\theta|\mathbf{D}) = P(\theta)\frac{P(\mathbf{D}|\theta)}{P(\mathbf{D})} \tag{1}$$

Vrijednost P(θ) nazivamo apriorna vjerojatnost klase, a uvjetnu gustoću P(D — θ) nazivamo klasom uvjetovana gustoća, odnosno izglednost klase.

U projekt smo uključili Weka biblioteku koja sadrži velik broj već unaprijed implementiranih algoritama strojnog učenja koji se mogu koristiti na skupu podataka, ili se metode mogu pozivati iz vlastitog Java koda. Mi smo odabrali korištenje iz vlastitog Java koda budući da smo napravili neke promjene koje su specifične za naš projekt.

3.1. Treniranje klasifikatora

Označili smo 173 tweeta od kojih je 39% bilo tweetova koji pripadaju kategoriji vijesti, 25% tweetova koji pripadaju kategoriji poslovnih ponuda i 36% tweetova koji pripadaju kategoriji privatnih poruka. Nakon toga smo proveli 10-struku unakrsnu provjeru (eng. 10-cross fold validation) koja služi za provjeru kako će se klasifikator ponašati na do sad nevidjenim primjerima. Kod desetorostruke unakrsne provjere skup podataka dijelimo na dva dijela: skup za učenje i skup za provjeru. Skup za učenje i skup za provjeru moraju biti disjunktni, odnosno ne smiju sadržavati iste primjere. Na skupu za učenje učimo model, a na skupu za provjeru provjeravamo njegovu generalizacijsku sposobnost. Ovaj postupak ponavljamo 10 puta, gdje svaki put uzimamo drugačiji skup za učenje i drugačiji skup za provjeru kako bismo dobili rezultate koji će najbolje odgovarati rezultatima na do sad nevidjenim primjerima.

Table 1: Preciznost, odziv i F1-mjera za kategoriju vijesti

Preciznost	Odziv	F1-mjera
0.98229	0.96500	0.95479

Table 2: Preciznost, odziv i F1-mjera za kategoriju poslovnih ponuda

Preciznost	Odziv	F1-mjera
0.84166	0.59166	0.68448

4. Mjerenja i rezultati

Za procjenu rada klasifikatora koristili smo preciznost, odziv i F1-mjeru. Za svaku kategoriju smo posebno izračunali svaku od ovih mjera [Tablice: 1, 2, 3].

Preciznost je udio točno klasificiranih primjera u skupu pozitivno klasificiranih primjera.

Odziv je udio točno klasificiranih primjera u skupu svih pozitivnih primjera.

F-mjera je harmonijska sredina preciznosti i odziva koju računamo koristeći (2).

$$F = 2 * \frac{Preciznost * Odziv}{Preciznost + Odziv} \tag{2}$$

5. Zaključak

Prvi korak u ovom radu bio je ujedno i jedan od najzahtjevnijih koraka, a to je pročišćavanje i priprema skupa podataka. Korisnici Twittera često koriste posebne znakove koje parseri ne prepoznaju te mogu vrlo lako narušiti performanse klasifikatora.

Klasifikacija tweetova jest zahtjevan zadatak. Iz samih tweetova, često je vrlo teško klasificirati tweet u odredjenu kategoriju te je stoga potrebno vrlo pažljivo odabrati značajke koje će se koristiti prilikom klasificiranja. Koristeći naivni Bayesov klasifikator, ostvarili smo vrlo dobru generalizaciju te izbjegli prenaučenost klasifikatora.

Klasifikacijom tweetova u jednu od tri kategorije omogućavamo korisniku da dohvaća samo najrelevantnije tweetove za njega, te izbjegne prebiranje po "šumi", njemu beskorisnih, tweetova.

Table 3: Preciznost, odziv i F1-mjera za kategoriju privatnih poruka

Preciznost	Odziv	F1-mjera
0.76753	0.87047	0.81365

Literatura

Ramanthan K. Bannerjee, S. and A. Gupta. Clustering short text using wikipedia. pages 861–862, Amsterdam, The Netherlands, July, 2007.

Sun N. Zhang C. Hu, X. and T. Chua. Exploiting internal and external semantics for the clustering of short texts using world knowledge. pages 919–928, Hong Kong, China, November, 2009.

Predictive Typing System

Mihael Šafarić, Matija Šantl

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{mihael.safaric,matija.santl}@fer.hr

Abstract

In this paper, we survey two algorithms for smoothing models for language n-gram modeling, Witten-Bell and Kneser-Ney. We have implemented a simple text editor that uses these methods and then we investigate how factors such as training data size, training corpus and size of n-gram affect to the final results of these methods, which is measured through cross-entropy and perplexity. We found that these factors can significantly affect the performance of models, especially training data size. The results are better when the larger corpus is used and when the n-gram size is bigger.

1. Introduction

A language model is a probability distribution $p(s)$ over string s that describes how often the string s occurs as a sentence in some domain of interest. Language models provide a text entry system with the power to predict unseen text likely to be generated by a user and they are used in many tasks including speech recognition, optical character recognition, handwriting recognition, machine translation, and spelling correction. This enables people to create text in collaboration with machines, which improves efficiency and makes usage of electronic devices more accessible to disabled people.

The most commonly used language models are trigram models and they determine the probability of a word given the previous two words, $p(\omega_i|\omega_{i-2}\omega_{i-1})$. The simplest way to approximate this probability is to compute

$$p(\omega_i|\omega_{i-2}\omega_{i-1}) = \frac{c(\omega_{i-2}\omega_{i-1}\omega_i)}{c(\omega_{i-2}\omega_{i-1})} \quad (1)$$

where the $c(\omega_{i-2}\omega_{i-1}\omega_i)$ is the number of times the word sequence $\omega_{i-2}\omega_{i-1}\omega_i$ occurs in some corpus of training data and $c(\omega_{i-2}\omega_{i-1})$ is the number of times the word sequence $\omega_{i-2}\omega_{i-1}$ occurs. This probability is called the maximum likelihood estimate. Maximum likelihood estimate can lead to poor performance in many applications of language models, for example, if the correct transcription of an utterance contains a trigram that has never occurred in the training data, we will have $p_{ML}(\omega_i|\omega_{i-2}\omega_{i-1}) = 0$ which will cause the wrong prediction. To address this problem, a technique called smoothing was used. The term smoothing describes techniques for adjusting maximum likelihood estimate of probabilities to produce more accurate probabilities.

In this work, we have implemented a simple text editor with predictive typing which uses Witten-Bell or Kneser-Ney smoothing technique to predict the next word with the possibility of choosing which n-gram will be used in this process.

2. Methods

In this section, we survey two smoothing algorithms for n-gram models, Witten-Bell smoothing and Kneser-Ney smoothing.

The first smoothing algorithm we're going to describe is the Witten-Bell smoothing algorithm. Witten-Bell smoothing algorithm is a very simple technique that performs rather poorly (Chen and Goodman, 1999). Next, we describe the Kneser-Ney smoothing algorithm.

2.1. Witten-Bell smoothing

Witten-Bell smoothing was developed for the task of text compression. The nth-order smoothed model is defined recursively as a linear interpolation between the nth-order maximum likelihood and the $(n-1)th$-order smoothed model.

$$N_{1+}(w_{i-n+1}^{i-1}\cdot) = |w_i : c(w_{i-n+1}^{i-1}w_i) > 0| \quad (2)$$

is the number of unique words that follow the history ω_{i-n+1}^{i-1}. The notation N_{1+} is meant to evoke the number of words that have one or more counts. This is used to determine the parameters $\lambda_{\omega_{i-n+1}^{i-1}}$ for Witten-Bell smoothing:

$$1 - \lambda_{\omega_{i-n+1}^{i-1}} = \frac{N_{1+}(w_{i-n+1}^{i-1}\cdot)}{N_{1+}(w_{i-n+1}^{i-1}\cdot) + \sum_{wi} c(\omega_{i-n+1}^{i})} \quad (3)$$

which leads to the expression to calculate the probability of the next word

$$p_{WB}(\omega_i|\omega_{i-n+1}^{i-1}) = \frac{c(\omega_{i-n+1}^{i}) + N_{1+}(\omega_{i-1}^{i-n+1}\cdot)pWB(\omega_i|\omega_{i-n+2}^{i-1})}{\sum_{wi} c(\omega_{i-n+1}^{i}) + N_{1+}(\omega_{i-1}^{i-n+1}\cdot)} \quad (4)$$

The factor $1 - \lambda_{\omega_{i-n+1}^{i-1}}$ can be interpreted as the frequency with which we should use the lower-order model to predict the next word. We should only use the lower-order model when there are no counts for the given n-gram in the higher-order model (i.e. when we see a novel word for the history). Expression $N_{1+}(w_{i-n+1}^{i-1}\cdot)$ can be viewed as number of novel words we have seen after the history ω_{i-n+1}^{i-1} over the training set, and this is in fact the count assigned by Witten-Bell smoothing to the lower-order distribution.

2.2. Kneser-Ney smoothing

Kneser-Ney smoothing algorithm was introduced in 1995 as an extension of absolute discounting where the lower-order distributions that one combines with a higher-order distribution is built in a novel manner (Chen and Goodman, 1999).

Next we'll present the mathematical background of the Kneser-Ney smoothing algorithm.

Considering bigram models, we would like to select a smoothed distribution p_{KN} that satisfies the following constraint on unigram marginals for all w_i:

$$\sum_{w_{i-1}} p_{KN}(w_{i-1}w_i) = \frac{c(w_i)}{\sum_{wi} c(w_i)} \tag{5}$$

where the function $c(w_i)$ denotes the count of the word w_i in the given corpus. The left hand-side of this equation is the unigram marginal for w_i of the smoothed bigram distribution p_{KN}, and the right-hand side is the unigram frequency of w_i found in the given corpus.

As in absolute discounting where $0 \leq D \leq 1$, we assume that the model has the following form:

$$p_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{max(c(w_{i-n+1}^{i}) - D, 0)}{\sum_{w_i} c(w_{i-n+1}^{i})} + \frac{D}{\sum_{w_i} c(w_{i-n+1}^{i})} N_{1+}(w_{i-n+1}^{i-1}\cdot) p_{KN}(w_i|w_{i-n+2}^{i-1}) \tag{6}$$

where

$$N_{1+}(\cdot w_i) = |w_i : c(w_{i-1}w_i) > 0| \tag{7}$$

is the number of different words w_{i-1} that precede w_i in the given corpus.

We used this formulation, because as stated in (Chen and Goodman, 1999), it leads to a cleaner derivation of essentially the same formula; no approximations are required, unlike in the original derivation.

By applying the law of total probability, we can write equations given above as following:

$$p_{KN}(w_{i-1}w_i) = \sum_{w_{i-1}} p_{KN}(w_i|w_{i-1}) p(w_{i-1}) \tag{8}$$

which leads to:

$$\frac{c(w_i)}{\sum_{wi} c(w_i)} = \sum_{w_{i-1}} p_{KN}(w_i|w_{i-1}) p(w_{i-1}) \tag{9}$$

Taking into account that $p(w_{i-1}) = \frac{c(w_{i-1})}{\sum_{wi-1} c(w_{i-1})}$, we have

$$c(w_i) = \sum_{w_{i-1}} c(w_{i-1}) p_{KN}(w_i|w_{i-1}) \tag{10}$$

which, after substituting and simplifying leads to the following form:

$$c(w_i) = c(w_i) - N_{1+}(\cdot w_i) + D p_{KN}(w_i) \sum_{w_i} N_{1+}(\cdot w_i) \tag{11}$$

Generalizing to higher-order models, we have the final form for the word probability:

$$p_{KN}(w_i|w_{i-n+2}^{i-1}) = \frac{N_{1+}(\cdot w_{i-n+2}^{i})}{\sum_{w_i} N_{1+}(\cdot w_{i-n+2}^{i})} \tag{12}$$

3. Results

Implemented language models with Witten-Bell and Kneser-Ney smoothing algorithms were evaluated using measures of perplexity and cross-entropy on separate fractions of the test corpus. Finally, a Text editor with the predicting capabilities was developed.

The most common metric for evaluating a language model is the probability that the model assigns to test data, or the derivate measures of cross-entropy and perplexity (Chen and Goodman, 1999). For a test corpus *T* composed of the sentences $(t1, t2, ..., t_{l_T})$ we can calculate the probability of the test corpus $p(T)$ as the product of the probabilities of all sentences in the corpus:

$$p(T) = \prod_{k=1}^{l_T} p(t_k) \tag{13}$$

The cross-entropy $H_p(T)$ of a model $p(t_k)$ on corpus T is:

$$H_p(T) = -\frac{1}{W_T} \log_2 p(T) \tag{14}$$

where W_T is the length of the corpus T measured in words. This value can be interpreted as the average number of bits needed to encode each of the W_T words in the test corpus using the compression algorithm associated with model $p(t_k)$.

The perplexity $PP_p(T)$ of a model p is the reciprocal of the (geometric) average probability assigned by the model to each word in the test corpus T and is related to cross-entropy with the following equation:

$$PP_p(T) = 2^{H_p(T)} \tag{15}$$

Lower cross-entropies and perplexities are better.

We have created three test corpuses to evaluate implemented smoothing algorithms. The main difference between them is the size. Test corpus *large.txt* has $1,000$ sentences, *medium.txt* has 500 sentences and *small.txt* has 50 sentences. Smoothing algorithms were trained and evaluated on separate fractions of the corpus. The size of the evaluation set was 5% of the train set. Results of these experiments can be seen in Tables 1, 2 and 3.

Table 1: Entropy and perplexity for *small.txt* corpus

Method	n	Probability	Entropy	Perplexity
Kneser-Ney	2	$4.115 \cdot 10^{-11}$	0.685	1.608
Witten-Bell	2	$3.662 \cdot 10^{-14}$	0.743	1.675
Kneser-Ney	3	$6.181 \cdot 10^{-12}$	0.621	1.537
Witten-Bell	3	1.0	0.0	1.0
Kneser-Ney	4	$4.955 \cdot 10^{-9}$	0.459	1.375
Witten-Bell	4	1.0	0.0	1.0

Table 2: Entropy and perplexity for *medium.txt* corpus

Method	n	Probability	Entropy	Perplexity
Kneser-Ney	2	$1.489 \cdot 10^{-70}$	0.102	1.073
Witten-Bell	2	$8.639 \cdot 10^{-78}$	0.112	1.081
Kneser-Ney	3	$3.727 \cdot 10^{-63}$	0.091	1.065
Witten-Bell	3	0.039	0.002	1.001
Kneser-Ney	4	$4.663 \cdot 10^{-58}$	0.083	1.059
Witten-Bell	4	0.307	$7.496 \cdot 10^{-3}$	1.0

Table 3: Entropy and perplexity for *large.txt* corpus

Method	n	Probability	Entropy	Perplexity
Kneser-Ney	2	$5.345 \cdot 10^{-174}$	0.317	1.246
Witten-Bell	2	$3.712 \cdot 10^{-190}$	0.347	1.272
Kneser-Ney	3	$3.142 \cdot 10^{-110}$	0.201	1.1.149
Witten-Bell	3	$7.834 \cdot 10^{-46}$	0.083	1.059
Kneser-Ney	4	$5.874 \cdot 10^{-114}$	0.207	1.155
Witten-Bell	4	$1.763 \cdot 10^{-11}$	0.019	1.014

4. Conclusion

If we take a closer look to the results, we can see that the implemented smoothing algorithms perform better when they have more context in prediction (size of the *n-gram*, n). However, the problem that we've encountered while evaluating smoothing algorithms was the low probability causing problems while calculating entropy and perplexity. Because of that and big run time complexity, the corpus size does not exceed $1,000$.

Text prediction enables text entry under difficult conditions, which is great for disabled users who have an opportunity to generate more text with fewer actions. There are two objections to predictive typing systems. The first is that computers will always have a limited ability to predict and the second is that people will gradually forget how to write text correctly.

Smoothing is a fundamental technique for statistical modeling, important not only for language modeling, but for many other applications as well. Whenever data sparsity is an issue, which is almost always in statistical modeling, smoothing can help increase performance.

References

Stanley F Chen and Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393.

Kumiko Tanaka-Ishii. 2010. Language models for text entry 2. *Text Entry Systems: Mobility, Accessibility, Universality*.

Classification and Grading of Sentiment in User Reviews

Edi Smoljan, Marko Deak, Tomislav Zoričić

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{edi.smoljan,marko.deak,tomislav.zoricic}@fer.hr

Abstract

The aim of this study is to build a basic model able to perform sentiment classification and rating prediction for user reviews of certain product groups. We use machine learning models based on support vector machines for both sentiment classification and rating prediction, with the features being extracted from the given user review datasets using feature selection based on mutual information and several NLP methods such as tokenization, stop word removal, stemming and TF-IDF scores. Different models for classification and prediction are cross-validated and compared among themselves using standard classification and regression performance measures and results are presented in both textual and graphical format. The results and findings can be used as a baseline for further development of a sentiment analysis pipeline able to perform better on reviews from different domains.

1. Introduction

The main focus of this study is sentiment analysis of user review using the datasets provided in (Blitzer et al., 2007) which consist of user reviews split into 4 domains: books, DVDs, electronics and kitchen and houseware products. The basic task of sentiment analysis is determining the polarity of a given text at the document/sentence/review level. This means that the model which performs this task should be able to discern between texts expressing *POSITIVE* or *NEGATIVE* opinions. This best translates to simple classification problems in the domain of machine learning and the approach is described in more detail in Section 3.

The second task of the sentiment analysis model which handles user reviews, as the one developed in the scope of this project, is prediction of the grades users left together with the review. In this example grades are integers in the range of $[1, 5]$ and this task is handled as a regression problem from the machine learning domain and is described in detail in Section 4.

In both tasks we use machine learning models, but to be able to do this, there is a need to build feature vectors from reviews. This preprocessing uses NLP methods, as described in Section 2., and also selects which features to actually use for later training and evaluation of classification and regression models. This is why sentiment analysis is considered a NLP task as the machine learning models are only a layer of the complete pipeline.

2. Review preprocessing

The first step towards classification of user reviews is creating the training dataset. This step is called preprocessing. The algorithm includes several steps, such as tokenization, removing stop words, tokens stemming, calculating *TF-IDF* and feature selection. The wanted preprocessing output consists of two files: a file containing the dictionary of words with their frequency, and second file containing *TF-IDF* weights for each word from dictionary for every review.

2.1. Algorithm

Every review record consists of a title, review text and its rating. Firstly, reviews are being parsed and retrieved from a selected set of review files and saved into an appropriate data structure in the program.

After that, all reviews need to be tokenized and all stop words removed, which is done by using methods from Apache Lucene. During tokenization process, all tokens go through extra controls of word, to "tidy up" potential mistakes remained after using Lucene methods. Extra controls consist of 4 checks : additional stop-word removal, which has a list of frequent words which don't give any semantic meaning to the sentence, and if the targeted word is amongst them then it's being removed; additional tokenizer which separates words connected by some special sign, like dot or exclamation; special sign trimmer which removes left over special signs from the start and end of the targeted word and lastly natural number remover which removes series of natural numbers, because they are not useful in this particular problem.

Stemming is the algorithms next part, which consists of one iteration through tokenized reviews while stemming the words using a Porter stemmer implementation and also counting words and their frequencies for dictionary creation. During the stemming process, we also have extra controls of input words which help us with the removal of unwanted tokens. Extra controls consist of 2 checks: removing words that have only one character or less, since they don't provide useful information and removing special signs from the ends of a input word, which is useful when after stemming the stemmed words have special signs on their end.

Also, we count appearances of bigrams and save them for extraction of most frequent bigrams, since they can have an impact on review classification. A subset of saved bigrams is added to the lexical feature set of instances based on parameters given by the user.

Finally, we can calculate the weight of features in documents. Calculation of weight consists of calculating term frequency, and inverse document frequency, as seen in for-

mula (1) below

$$w_{ij} = tf(k_i, d_j) \cdot idf(k_i, D) \tag{1}$$

TF (term frequency) is calculated as relative term frequency, i.e. frequency of target word dividied by frequency of most frequent word as shown in formula (2).

$$tf(k_i, d_j) = 0.5 + \frac{0.5 \cdot freq(k_i, d_j)}{max(freq(k, d_j)|k \epsilon d_j)} \tag{2}$$

IDF (inverse document frequency) shown in formula (3) is logarithm of total number of documents divided by number of documents in which the target word appears.

$$idf(k_i, D) = log \frac{|D|}{|(d \epsilon D | k_i \epsilon d_j)|} \tag{3}$$

Another part of the preprocessing algorithm is the feature selection. Feature selection is done by using previously created dictionary and weight calculation result. Features are firstly ranked by the mutual information criteria which is described by the formula (4) where U is random variable that takes e_t = 1 (the document contains term t) and e_t = 0 (the document does not contain t), and C is a random variable that takes values e_c = 1 (the document is in class)and e_c = 0 (the document is not in the class).

$$I(U;C) = \sum_{e_t \epsilon \{1,0\}} \sum_{e_c \epsilon \{1,0\}} P\left(U = e_t, C = e_c\right) log_2 \frac{P\left(U = e_t, C = e_c\right)}{P(U = e_t)(P = e_c)} \tag{4}$$

For MLEs of the probabilities, upper formula is equivalent to equation (5) where N's are counts of documents that have the values of e_t and e_c that are indicated by the two subscripts. This is the formula used in our calculation. After calculating the mutual information value for every class and every feature, the feature list is sorted by the sum of calculated values for every feature. A fraction of this sorted list of feature is then used as the final feature set.

$$I(U;C) = \frac{N_{11}}{N} log_2 \frac{N N_{11}}{N_{1.} N_{.1}} + \frac{N_{01}}{N} log_2 \frac{N N_{01}}{N_{0.} N_{.1}} + \frac{N_{10}}{N} log_2 \frac{N N_{10}}{N_{1.} N_{.0}} + \frac{N_{00}}{N} log_2 \frac{N N_{00}}{N_{0.} N_{.0}} \tag{5}$$

After calculation, the feature list (dictionary) and *TF-IDF* weighted reviews are stored in separate files.

3. Classification

3.1. Classification problem

The original task that we were given was to train a classifier that assigns one label to a previously unseen review. The reviews had to be assigned with one of the labels from the set *POSITIVE*, *NEUTRAL* and *NEGATIVE*. Every review of a product comes with a score from 1 to 5 of that product so the first idea was to assign the label *POSITIVE* to the reviews with the score of 4 or 5, *NEGATIVE* to the reviews with the score of 1 or 2 and lastly *NEUTRAL* to the remaining reviews with the score of 3. However, the dataset used for training described in (Blitzer et al., 2007). that we were provided with has no reviews scored with 3, so an alternative was to label the reviews scored with 2 and 4 as *NEUTRAL*. Another alternative is to completely exclude the *NEUTRAL* class and reduce the problem to a binary classification one. Results of both alternatives are presented in this section.

All classifiers in this section are trained with the help of *LIBLINEAR* (Fan et al., 2008), a program for training linear classifiers which also provides tools for automated cross-validation of a selected parameter. The initial review set is divided into a training set and a test set in ratio of $4:1$. It has been shown in (Hsu et al., 2003) that better results are obtained if the initial parameters are scaled to a range from $[-1, 1]$ or $[0, 1]$ so the entire review set is scaled to a range from $[0, 1]$. A grid search algorithm which searches for the best parameter in term of cross-validation accuracy on the train set with 5 folds has been implemented. The search range for the optimal parameter in the grid search algorithm is from 2^{-10} to 2^{10} with the exponent step of 1. The review set for every domain in this text has been obtained by parsing only the *negative.review* and *positive.review* files in their respective domain folders because those files have a pretty much balanced set. We chose L_2 regularized logistic regression as our classifier model.

3.2. Simple training

In this section the results of classifiers trained in a way described in section 3.1. on different domains are presented. Table 1 shows best values of accuracy, macro precision, macro recall, macro F1 score and the hiperparameter value for which these scores were obtained on the test set in case of binary classification. Table 2 shows the values of the same scores but in this case the scores are obtained by a 3-class classifier.

Table 1: Scores for simple trained binary classification

	C	Acc	P^M	R^M	F_1^M
Books	1.000	0.798	0.798	0.798	0.797
DVD	0.250	0.845	0.847	0.845	0.845
Electronics	1.000	0.855	0.856	0.855	0.855
Kitchen	0.250	0.875	0.875	0.875	0.875

Table 2: Scores for simple trained 3-class classification

	C	Acc	P^M	R^M	F_1^M
Books	0.250	0.618	0.619	0.614	0.615
DVD	0.031	0.650	0.657	0.638	0.643
Electronics	0.063	0.673	0.668	0.671	0.668
Kitchen	0.063	0.688	0.659	0.658	0.645

3.3. Training with bigrams

Using bigrams in addition with single words is a popular method of improving classifier performance as shown in (Tan et al., 2002). To enhance the performance of the classifier trained in Section 3.2. a subset of all bigrams in the

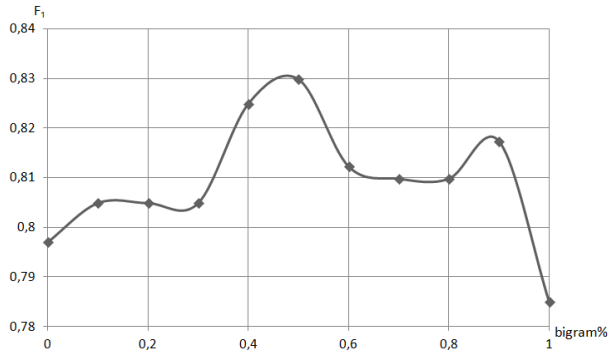


Figure 1: Graph of F_1^M dependency in terms of additional bigram percentage

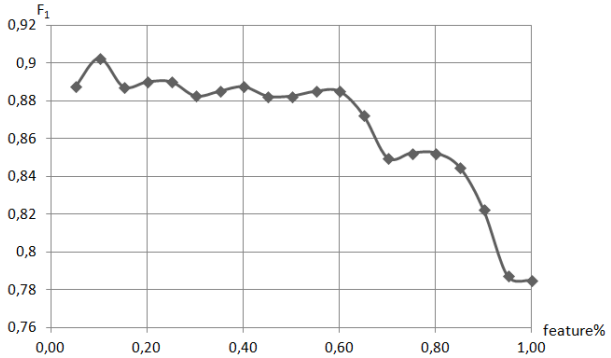


Figure 2: Graph of F_1^M dependency in terms of selected feature percentage

review set is taken into account. The mentioned subset is constructed by removing all bigrams which appear less than 3 times in the review set. The amount of bigrams which will be added in the feature set is optimized in a way that the classifier trained on the train set in a way described in the previous section 3.2. minimizes the F_1^M score on the test set. Table 3 shows best values of accuracy, macro precision, macro recall, F_1^M score and the percentage of bigrams for which these scores were obtained on the test set in case of binary classification, table 4 shows the same thing in the case of 3 class classification.

Table 3: Scores for binary classification trained with bigrams

	bigram%	Acc	P^M	R^M	F_1^M
Books	50%	0.830	0.831	0.830	0.830
DVD	20%	0.868	0.868	0.868	0.867
Electronics	60%	0.870	0.870	0.870	0.870
Kitchen	60%	0.878	0.878	0.878	0.877

Table 4: Scores for 3-class classification trained with bigrams

	bigram%	Acc	P^M	R^M	F_1^M
Books	30%	0.635	0.644	0.627	0.632
DVD	50%	0.648	0.650	0.641	0.644
Electronics	50%	0.673	0.666	0.671	0.664
Kitchen	10%	0.698	0.676	0.678	0.675

The graph of different F_1^M scores for different values of additional bigrams percentage of a classifier trained on the book reviews set is shown on figure 1.

3.4. Training with bigrams and additional feature selection

Feature selection is another popular method of boosting classifier performance as shown in (Manning et al., 2008). In the initial feature list the entire subset of bigrams constructed in a way that has been shown in Section 2.1. is added and the feature selection based on Mutual information described in section 2.1. is conducted. The features are ranked by the sum of mutual information for every class and then a percentage of the best ranked features is selected for the final feature set. The percentage of features to be taken in the final feature set is optimized in a way that the classifier trained on the train set in a way described in section 2.1. minimizes the F_1^M score on the test set. Table 5 shows best values of accuracy, macro precision, macro recall, F_1^M score and the percentage of best ranked features for which these scores were obtained on the test set in case of binary classification, table 6 shows the same thing in the case of 3 class classification.

Table 5: Scores for binary classification trained with bigrams and feature selection

	feature%	Acc	P^M	R^M	F_1^M
Books	10%	0.903	0.903	0.903	0.902
DVD	10%	0.910	0.910	0.910	0.910
Electronics	65%	0.888	0.888	0.888	0.887
Kitchen	65%	0.918	0.918	0.918	0.917

Table 6: Scores for 3-class classification trained with bigrams and feature selection

	feature%	Acc	P^M	R^M	F_1^M
Books	10%	0.763	0.771	0.761	0.765
DVD	25%	0.770	0.772	0.769	0.770
Electronics	25%	0.7425	0.741	0.741	0.737
Kitchen	20%	0.778	0.763	0.761	0.760

The graph of different F_1^M scores for different values of best ranked features percentage of a classifier trained on the book reviews set is shown on figure 2.

4. Rating prediction in user reviews

The last component of the system developed in the scope of this project uses regression methods from machine learning to predict the ratings given in user reviews. The same set used for classification was used again to both train and test

models, with the added difference that this time the training outputs were not the sentiments themselves, but simply grades the users left with the reviews. This time the data set was split so that 70% of the data was used as the training set, and 30% as the test set. The rating prediction models developed in the project were evaluated using mean absolute error (MAE) and the Pearson correlation coefficient (r) on the test set as measures. These are defined as

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|f_i - y_i| \tag{6}$$

$$r = \rho_{F,Y} = \frac{E[(f-\mu_f)(x-\mu_x)]}{\sigma_f \sigma_x} \tag{7}$$

where f_i is the prediction of the model for the i-th set of input features, and y_i is the expected i-th output.

4.1. Artificial neural network model

The initial approach used was using artificial neural networks evolved with genetic algorithms. This approach was abandoned rather quickly as training (evolution to be more precise) of the model took more time than the 1 minute that was set as the training time upper boundary (this was considered a reasonable boundary as it was measured on the *BOOK* subset on which preprocessing takes around 2 seconds and the other two models are trained in less than 10 seconds), while in the same time giving MAE higher than 1 which meant that the model was giving predictions that were off by a grade on average. As this meant the model was off by 20% of the domain, this was a level of error that was clearly not acceptable considering the time it took to train.

4.2. SVR model

The second approach consisted of using support vector machines extended for regression problems (support vector regression or SVR) using the RBF kernel. The variant we used was the ϵ-SVR, where parameters C and ϵ of the support vector machine and the parameter γ of the RBF kernel were optimized using 5-fold cross-validation. Model training, testing and validation were done using the *LIBSVM* library, further described in (Chang & Lin, 2011)

4.3. Linear regression model

As the ratio of data samples available to the number of features was rather high, the last approach that we theorized to be able to perform well and that was tested uses SVMs with the linear kernel and it can be viewed as a linear regression model as there is no explicit or implicit transformation to higher dimensions. The reason for our initial expectations of the model's performance was due to the number of features being high enough for there to be a possibility of finding a good enough separating hyperplane in the feature space.

L_2 regularization was used in training as it was shown in a number of works to perform better than the L_1 regularization given enough data samples compared to the number of features. As in Section 4.2., we again used the ϵ-SVM variant of the support vector machine model, but as this time the linear kernel was used, the only parameters which needed to be optimized were C and ϵ of the SVM and this was again done using 5-fold cross-validation. Model training, testing and validation were done using the *LIBLINEAR* library which is further described in (Fan et al., 2008).

4.4. Performance

The performance of the models is shown in Tables 7 and 8 below, where the performance of each model with its best parameters (the ones found by cross-validation) is shown. From the tables, it can be noticed that the model using support vector regression with the RBF kernel outperforms the linear one by a margin of around 0.05 (considering MAE) which is significant to a certain extent. However, it must be noted that both models actually perform extremely well on the given data, and if their outputs were rounded up to the nearest integers (as review grades can only be integers in this specific problem), the MAE in both cases would actually fall to zero.

Table 7: Scores for the linear regression model trained with bigrams

	bigram%	MAE	r
Books	50%	0.8086	0.8095
DVD	40%	0.8451	0.7997
Electronics	30%	0.8323	0.8205
Kitchen	40%	0.8330	0.8191

Table 8: Scores for the support vector regression model trained with bigrams

	bigram%	MAE	r
Books	40%	0.7086	0.8421
DVD	30%	0.7456	0.8455
Electronics	30%	0.7278	0.8318
Kitchen	40%	0.7359	0.8262

Furthermore, we also tried using the most frequent bigrams as features, as we did before with classification. On figure 3 given below we can see how the mean absolute error of each model changes when a certain percentage of most frequent bigrams is added to the set. At first the mean absolute error slightly decreases as more bigrams are added, but it must be noted that there is a point where adding more bigrams to the feature vector actually hinders the performance of models, especially the SVR model using the RBF kernel. This is most probably a problem of overtraining, to which the linear model is expected to be less susceptible as it is simpler and therefore is more probable to be able to generalize well. The results in the graph below were all calculated on the *BOOK* subset of the data. The higher and darker line represents the linear regression model, while the lower line represents the SVR model with the RBF kernel.

The last thing to note is that, unlike the first approach with neural networks, with these models we ended up using feature selection algorithms based on mutual information. This reduces the number of features allowing us to build

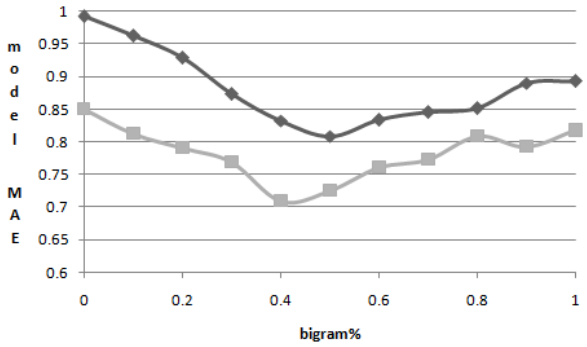


Figure 3: Graph of MAE dependency in terms of additional bigram percentage

simpler models and could eventually lead to the initial approach with artificial neural networks trained using genetic algorithms being used again, but it was not needed to be used in the scope of this project as both models described managed to outperform it in means of MAE and Pearson correlation coefficient while also needing less time to be trained, cross-validated and tested.

5. Conclusion

In the case of classification we have shown that inclusion of additional lexical features in terms of bigrams in most cases slightly improves classification performance. However, the biggest performance gain is achieved with feature selection which can in some cases improve classifier F1 score by 7% which shows that classifiers built with feature selection are more capable of better generalization.

With all the additional ways of improving performance we managed to construct classifiers that have F_1^M scores of about 0.9 and accuracy of about 90% in the case of binary classification and F_1^M scores of about 0.75 and accuracy of 76% in the case of 3 class classification.

From the classifiers we trained, we managed to extract a list of words (and bigrams) which have the greatest influence on the predictions the classifier makes. These words are called discriminative words and could potentially be used to develop a sentiment lexicon that could be used in developing features for more advanced models. They can be split into words that make the classifier decide that a review is positive and those that make the classifier classify a review as negative. For example, in the *ELECTRONICS* subset, the 10 most positive words/bigrams are: *great, excel, perfect, best, price, good, easi, highli, fast and highli recommend*(bigram), while the 10 most negative words are: *return, disappoint, poor, bad, was, back, terribl, unfortun, try and monei wast*(bigram). Of course, all of these are stemmed and specific for the *ELECTRONICS* domain (even though some of them may be discriminative words in other domains), which was chosen for demonstration here simply for the fact that the bigrams are clearly shown in the list of discriminative words.

In the case of rating prediction, we managed to develop models with MAE of about 0.7086 and Pearson correlation coefficient of about 0.8421 when using support vector regression with the RBF kernel, and MAE of about 0.8086 and Pearson correlation coefficient of about 0.8095 when using simple support vector regression with the linear kernel, which can be considered linear regression.

In both sentiment classification and rating prediction, we managed to reach satisfactory results, and further improvement would need to focus more on the system being able to perform well in multiple domains (as opposed to one model per domain approach that was used in this study) rather than trying to develop a system with better classification or regression performance for the same problem.

References

John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. 2007.

Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

Chade-Meng Tan, Yuan-Fang Wang, and Chan-Do Lee. The use of bigrams to enhance text categorization. In *INF. PROCESS. MANAGE*, pages 529–546, 2002.

Chih wei Hsu, Chih chung Chang, and Chih jen Lin. A practical guide to support vector classification, 2003.

Predictive typing pomoću Kneser-Ney modela

Tomislav Tenodi, Juraj Selanec, Luka Soldo

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{tomislav.tenodi,juraj.selanec,luka.soldo}@fer.hr

Sažetak

Kneser-Neyevo izgladivanje predstavlja jezični model koji se koristi za izračunavanje vjerojatnosti pojavljivanja n-grama unutar nekog korpusa. U ovom radu se koristi u sklopu trigramskih i bigramskih modela u svrhu izgradnje predictive typing sustava. Model je i evaluiran preko različitih parametara kao što su perplexity, cross-entropy i memorijsko zauzeće te je pokazana razlika izmedu bigramskih i trigramskih modela. Prikazana je i teorijska osnova Kneser-Neyevog izgladivanja za bigrame i trigrame, njegova evaluacija na dijelu početnog korpusa te sama implementacija unutar .NET okoline. Sama aplikacija sadrži funkcionalnosti učitavanja korpusa, njegove analize, evaluacije Kneser-Neyevog modela za bigrame/trigrame te sustav za predvidanje korisnikovih unosa koji koristi backoff na (N-1)-gram model.

1. Uvod

U ovom projektu zadatak je bio napraviti predictive typing sustav, čija je glavna zadaća (na temelju teksta koji je sustav prije dobio i analizirao) predvidanje slijedeće riječi koje unosi korisnik. Sam sustav sadrži funkcionalnost učitavanje korpusa te njegovu analizu iz koje se dobivaju podaci potrebni za sljedeće korake. Nakon analize, odnosno prikupljanja podataka, sustav modelira jezik pomoću kojeg vrlo uspješno predvida korisnikov unos. Modeliranje jezika omogućuje odredivanje pripada li proizvoljan tekst odredenom jeziku. Jedan tip modeliranja jezika je probabilističko modeliranje koje koristi vjerojatnosti da procijeni pripada li odredeni dio teksta nekom jeziku, te je upravo taj tip modeliranja bio korišten u našem predictive typing sustavu (Chen and Goodman, 1999).

Koristeći probabilističko modeliranje prilikom izrade našeg predictive typing sustava, naišli smo na problem koji se može jednostavno predočiti sljedećim primjerom. Recimo da se u tekstu često pojavljuje 'San Francisco' gdje se riječ 'Francisco' pojavljuje samo poslije riječi 'San', no unatoč tome riječ 'Francisco' poprima veliku vjerojatnost pojavljivanja. Ukoliko se dobije sljedeći niz za nadopuniti: 'I can't see without my reading ...', postoji mogućnost da se sustav odluči za riječ 'Francisco' zbog velike vjerojatnosti pojavljivanja, iako svakoj osobi upoznatoj sa engleskim jezikom je jasno da je sljedeća riječ 'glasses'. Ova pojava definitivno nije bila dopustiva u našem sustavu a samo rješenje na taj problem nam daje Kneser-Neyevo izgladivanje. Kneser-Neyevo izgladivanje rješava taj problem na način da ne gleda vjerojatnost pojavljivanja unigrama u tekstu već gleda kolika je vjerojatnost pojavljivanja riječi w nakon odredenog konteksta bigrama ili trigrama. Kneser-Neyeva formula:

$$P_{KN}(w_i|w_{i-1}) = \frac{max(c(w_i|w_{i-1}) - d), 0)}{count(w_{i-1})} + L * P_{cont}(w) \quad (1)$$

$$L(w_{i-1}) = \frac{d}{c(w_{i-1})}|\{w : c(w_{i-1}, w) > 0\}| \quad (2)$$

$$P_{cont}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|} \quad (3)$$

Samu analizu korpusa čini jednostavno prebrojavanje pojave različitih unigrama te pojave odredenih bigrama i trigrama iz kojih su se gradile prikladne strukture podataka. Konkretno, radi se o strukturama Dictionary<string, int>za unigrame, Dictionary<string, Dictionary<string, int>>za bigrame, te takoder Dictionary<string, Dictionary<string, int>>za trigrame. Ključ u rječniku kod unigrama predstavlja sama riječ a vrijednost je broj pojavljivanje riječi. Kod bigrama i trigrama situacija je kompliciranija te se koristi rječnik kao vrijednost vanjskog rječnika. U slučaju bigrama ključ prvog rječnika je prva riječ u bigrama a ključ drugog rječnika je druga riječ u bigramu, dok kod trigrama prvi ključ predstavlja spojene prve dvije riječi. Takva struktura je odabrana radi bržeg pristupa podacima prilikom obavljanja same analize jer su pojedine riječi dostupne preko svojih ključeva. Kasnije se podaci iz struktura koriste za dinamičko generiranje slijedećeg unosa korisnika, koristeći upravo gore navedene formule, te za evaluaciju samo sustava. Dodatna mogućnost našeg sustava je i paralelna analiza korpusa koja omogućuje paralelno prebrojavanje pojave unigrama, bigrama i trigrama na više procesora što uvelike ubrzava trajanje same analize koristeći pritom MPICH2 implementaciju MPI standarda.

2. Pregled polja - slični algoritmi i sustavi

Jezični modeli su osnovni element NLP-a koji se koriste u širokim primjenama od spellcheckinga, prepoznavanja govora, POS označavanja, parsiranja itd... Prilikom pretraživanja informacija (information retrieval), svaki dokument u kolekciji ima jezični model kojeg dijeli (a i ne mora) s ostalim dokumentima. Kada korisnik postavi neki upit Q, jezični model pojedinog dokumenta će procijeniti koliko dobro taj dokument odgovara upitu Q. U praksi se najčešće koristi unigram model koji je dovoljno dobar za pretraživanje informacija na temelju klasificiranja jedne riječi unutar neke domene, tj. teme. Kod unigrama se izračunava vjerojatnost pojavljivanja odredene riječi bez dodatnog konteksta, tj. bez susjednih riječi koje prethode/slijede tu riječ. Kod bigrama se gleda trenutna riječ i riječ prije nje, tj. broj pojavljivanja tog bigrama podijeljen

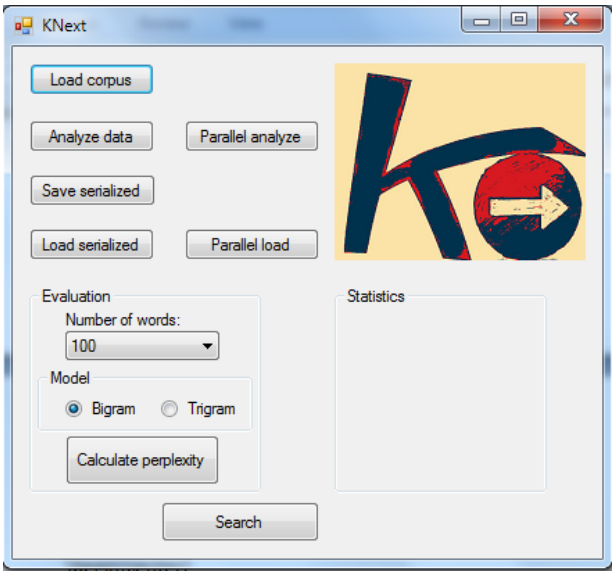


Figure 1: Slika početne forme.

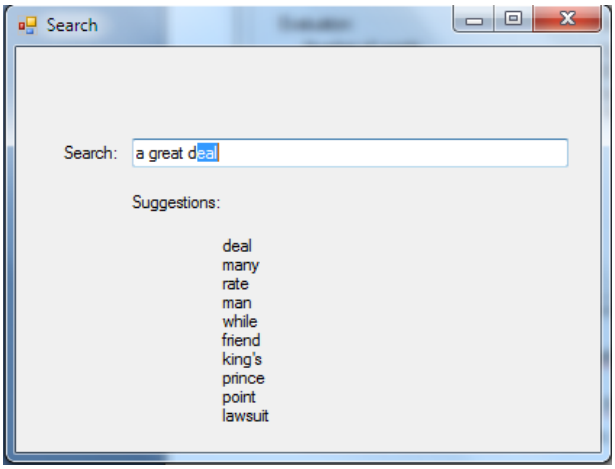


Figure 2: Slika forme u kojoj se predvida korisnikov unos.

s brojem pojavljivanja prve riječi (Jurafsky, 2013).

$$p(w_i|w_{i-1}) = \frac{count(w_i|w_{i-1})}{count(w_{i-1})} \quad (4)$$

Izračunavanje vjerojatnosti pojavljivanja niza riječi može biti težak problem jer kompliciране fraze ili rečenice ne moraju biti 'uhvaćene' prilikom treniranja jezičnog modela nego moraju biti uključene prilikom izgladivanja. Postoje različiti n-gram jezični modeli koji koriste izgladivanja, a onaj koji je korišten u ovom radu je Kneser-Ney. Additive smoothing je osnovni model čije su performanse loše, ali služi kao dobar polazni primjer za sve ostale modele izgladivanja. Ovo izgladivanje radi na principu da broju pojavljivanja svakog n-grama dodaje neku malu vrijednost alfa (obično 1) u brojniku, a u nazivniku (ukupni broj riječi) dodaje veličinu ukupnog vokabulara nekog korpusa pomnoženog s parametrom alfa. Na taj način i n-grami koje se pojavljuju 0 puta dobiju neku malu vjerojatnost pojavljivanja. Good Turing metoda premješta vjerojatnosnu masu za n-grame koji se pojavljuju r + 1 put u n-grame koji se pojavljuju r puta. Ova metoda je važna jer ju implementira (u odredenom dijelu) i Kneser-Neyevo izgladivanje. Jelinek Mercer metoda interpolira n-grame na temelju (n-1)-grama pa tako bigrami koji sadrže frekventnije unigrame i sami imaju veću vjerojatnost pojavljivanja. Witten Bell metoda modificira Jelinek Mercer i uz Kneser-Ney predstavlja state-of-the-art jezični model.

3. Opis modela

Kao što je već spomenuto, Good Turing metoda uzima dio vjerojatnosti pojavljivanja n-grama koji se pojavljuju (r+1) puta i daje ih n-gramima koji se pojavljuju r puta. Taj iznos se mora posebno izračunati za svaku frekvenciju, medutim vrijednosti su podjednake i iznose oko 0.75. Umjesto izračuna za svaku frekvenciju pojavljivanja, uzet će se fiksna vrijednost d u iznosu od 0.75 kao interpolacija. Takvu interpolaciju koristi i Kneser-Ney (bigramski model):

$$P_{KN}(w_i|w_{i-1}) = \frac{max(c(w_i|w_{i-1}) - d), 0)}{c(w_{i-1})} + L * P_{cont}(w) \quad (5)$$

Prikazana je jednadžba Kneser-Neyevog modela za bigrame gdje je prvi član vjerojatnost odredenog bigrama s oduzetom vjerojatnosnom masom d.

U jednadžbi postoje 2 dodatna člana,

$$L \; i \; P_{cont}$$

. L predstavlja konstantu normalizacije, vjerojatnosnu masu d koju smo oduzeli:

$$L(w_{i-1}) = \frac{d}{c(w_{i-1})} |\{w : c(w_{i-1}, w) > 0\}| \quad (6)$$

U nazivniku se nalazi ukupan broj pojavljivanja prve riječi, te se taj razlomak množi brojim različitih bigrama u kojem se pojavljuje prva riječ. Kao što je već i spomenuto u uvodu, Dan Jurafsky dao je dobar primjer koji opisuje ponašanje zadnjeg člana Kneser-Neyevog jezičnog modela: Dovrši rečenicu: 'I can't see without my reading ...'. Ovaj put ćemo pogledati zašto je to tako. Rečenica bi trebala završavati s 'glasses'. Medutim, budući da je San Francisco bigram koji se često pojavljuje, neki drugi model će pokazati da on bolje odgovara od glasses. Kneser-Ney ovaj problem lagano riješava pitajući se u koliko različitih bigrama se druga riječ (Francisco) nalazi.

$$P_{cont}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|} \quad (7)$$

Kneser-Neyeva jednadžba za n-grame koji su veći u 2 je rekurzivna uz manje modifikacije. Prvi član za poziv Kneser-Neya najvišeg reda ostaje isti, dok niži članovi odgovaraju Pcont višeg reda. Primjerice u trigramima je pri pozivu KN za trigram prvi član jednak kao u prethodnom primjeru, dok je u rekurzivnom pozivu za bigrame jednak Pcont za trigram.

Table 1: Statistike korpusa

Parametar	Broj
Broj riječi	83000
Veličina rječnika	18000

$$P_{KN}(W_i|w_{i-n+1}^{i-1}) = \frac{max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \quad (8)$$

$$L(w_{i-n+1}^{i-1}) * P_{KN}(w_i|w_{i-n+2}^{i-1}) \quad (9)$$

Pri implementaciji se koristio trigramski Kneser-Ney s backoffom na bigramski.

4. Evaluacija modela

Model smo evaluirali preko 3 parametra - perplexity, cross-entropy i memorijsko zauzeće. Statistike jezičnog korpusa na kojemu je treniran model je prikazan u tablici 1.

Perplexity je mjera koliko dobro naš model predvida uzorak unutar korpusa i dan je izrazom (Jurafsky, 2013a):

$$PP(W) = P(w_1 w_2 w_3 ... w_N)^{\frac{-1}{N}} \quad (10)$$

odnosno:

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1 w_2 w_3 ... w_N)}} \quad (11)$$

gdje je N broj riječi u uzorku. Vjerojatnosti pojavljivanja svake riječi se medusobno pomnože, invertiraju i normaliziraju N-tim korijenom. Iz jednadžbe je vidljivo da što je perplexity manji, model je bolji. Cross-entropy je povezan s perplexityjem:

$$CE(w) = log_2(PP(w)) \quad (12)$$

U tablici 2. je prikazan perplexity za bigramske i trigramske modele na uzorcima od 100, 1000, 10000 i 83000 riječi korpusa na kojem je model i treniran. Vidljivo je da je trigramski model mnogo uspješniji od bigramskog što je i razumljivo budući da pamti veći broj riječi. Takoder se vidi i da perplexity ne raste nakon 1000. riječi što znači da se stabilizirao. Cross-entropy je usko povezan s perplexityjem pa i za njega vrijedi isto. Prikazan je na tablici 3. Ukoliko se model trenira na manjem dijelu korpusa, rezultati mogu biti ili gori ili bolji, ovisno o tome koji skup je uzet za izračunavanje perplexityja. Ako se radi o manjem skupu, onda bi i perplexity mogao biti bolji jer je model treniran na tekstu koji bolje odgovara testnom setu (Maccartney, 2005;Gawron, 2013).

Korištenje memorije je prikazano u tablici 4. gdje je vidljivo da je u početku zauzeće memorije 100 puta veće od jezičnog korpusa, ali ta brojka padne na otprilike 7 jer pojavljivanje novih riječi prestane te se pojavljuju samo stare (dictionary struktura podataka). Rast je u tom trenutku sublinearan, ali bi mogao doći čak i do logaritamskog.

Table 4: Memorijski otisak

Veličina teksta (MB)	Veličina analiziranog teksta (MB)
500 KB	44
13	150
26	230
52	400
104	720

5. Zaključak

Kneser-Neyev jezični model je state-of-the-art model koji riješava probleme koje drugi modeli nisu uspjeli riješiti. U ovom radu je prikazano kako je on implementiran unutar predictive typing sustava i kao takav pruža fenomenalne rezultate koji korisniku olakšavaju rad.

Iskorišten je unutar trigram modela koji ima mogućnost backoffa na bigram ako upit ne postoji. Isto tako, analiza korpusa je dovoljno brza da omogućava znatno proširivanje novim tekstovima i efikasnije korištenje sustava. Mogućnost napretka se vidi u efikasnijem raspolaganju memorijom koja se koristi u prevelikim količinama i kao takva predstavlja prepreku kod proširivanja modela.

Osim toga, omogućeno je automatsko evaluiranje modela i za bigrame i za trigrame, automatsko učitavanje i analiza korpusa kao i grafičko sučelje koje korisniku omogućava jednostavno i zanimljivo korištenje predictive typinga.

Literatura

S. F. Chen and J. Goodman, *An empirical study of smoothing techniques for language modeling*, Computer Speech and Language, 1999.

Dan Jurafsky, *Kneser-Ney Smoothing*, https://class.coursera.org/nlp/lecture/20, 2013., Online: accessed 9-June-2014

Dan Jurafsky, *Perplexity and evaluation*, https://class.coursera.org/nlp/lecture/129, 2013., Online: accessed 9-June-2014

Bill Maccartney, *NLP lunch tutorial: Smoothing*, http://nlp.stanford.edu/ wcmac/papers/20050421-smoothing-tutorial.pdf, 2005., Online: accessed 9-June-2014

Jean Mark Gawron, *Discounting*, http://www-rohan.sdsu.edu/ gawron/compling/course-core/lectures/kneser-ney.pdf, 2013., Online: accessed 9-June-2014

Table 2: Perplexity za n-gramske modele

Broj riječi	Trigrami	Bigrami
100	7.66	43.81
1000	8.92	65.32
10000	9.16	68.82
83000	9.45	68.37

Table 3: Cross-entropy za n-gramske modele (bitova po riječi)

Broj riječi	Trigrami	Bigrami
100	2.93	5.45
1000	3.15	6.02
10000	3.19	6.1
83000	3.24	6.09