COMENIUS UNIVERSITY, BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

Improving LSA word weights for document classification

master's thesis

2018
Bc. Vladimír Macko

COMENIUS UNIVERSITY, BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

IMPROVING LSA WORD WEIGHTS FOR DOCUMENT CLASSIFICATION

MASTER'S THESIS

Study programme:	Informatics
Study field:	Informatics
Department:	Department of Computer Science
Supervisor:	RNDr. Kristína Malinovská, PhD.
Consultant:	RNDr. Radim Řehůřek, PhD.

Bratislava, 2018
Bc. Vladimír Macko

87659972

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Vladimír Macko
Študijný program:	informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Improving LSA word weights for document classification
Adaptácia váh slov v LSA pre zlepšenie klasifikácie dokumentov

Anotácia: V NLP momentálne dominujú prístupy založené na predikcii (napr. Word2Vec) [1]. V práci sa vraciame k známemu prístupu na báze kookurencií slov, k latentnej sémantickej analýze (LSA) [2]. Keďže LSA reprezentuje dokumenty cez najreprezentatívnejšie črty a nie cez najdiskriminatívnejšie, môže zlyhávať pri klasifikačných úlohách. Podobne ako v slovných váhach s učiteľom [3], aj LSA možno adaptovať aby sa učilo špecifické váhy slov pre konkrétne klasifikačné úlohy pomocou gradientového učenia.

Cieľ: 1. Navrhnúť nový spôsob učenia váh slov s učiteľom pre konkrétny problém.
2. Vyhodnotiť úspešnosť navrhovaného prístupu pomocou experimentov.
3. Porovnať navrhovaný prístup s inými (bežnými) váhovými schémami.

Literatúra: [1] Baroni, M., Dinu, G. and Kruszewski, G., 2014. Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In Proceedings of 52nd Ann. Meeting of the ACL, Vol. 1, pp. 238-247.
[2] Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K. and Harshman, R., 1990. Indexing by latent semantic analysis. Journal of the American society for information science, 41(6), p.391.
[3] Deng, Z.H., Luo, K.H. and Yu, H.L., 2014. A study of supervised term weighting scheme for sentiment analysis. Expert Systems with Applications, 41(7), pp.3506-3513.

Kľúčové slová: spracovanie prirodzeného jazyka, klasifikácia dokumentov, LSA, gradient descent

Vedúci:	RNDr. Kristína Malinovská, PhD.
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	prof. Ing. Igor Farkaš, Dr.
Dátum zadania:	09.04.2018

Dátum schválenia: 11.04.2018 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

....

študent vedúci práce

87659972

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Vladimír Macko
Study programme:	Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science, Informatics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Improving LSA word weights for document classification

Annotation: NLP is currently dominated by prediction-based approaches (e.g. Word2Vec) [1]. Here we revisit a famous co-occurrence-based approach, the Latent Semantic Analysis (LSA) [2]. LSA finds the most representative rather than the most discriminative features of documents and hence may perform poorly in document classification. Inspired by the work on supervised term weights [3], the LSA can be enhanced to learn task specific weights using gradient descent.

Aim: 1. Propose a novel approach to learning task-specific word weights using gradient descent.
2. Explore the performance of this new approach using experimental approach.
3. Compare the new approach with other (commonly used) weighting schemes.

Literature: [1] Baroni, M., Dinu, G. and Kruszewski, G., 2014. Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (Vol. 1, pp. 238-247).
[2] Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K. and Harshman, R., 1990. Indexing by latent semantic analysis. Journal of the American society for information science, 41(6), p.391.
[3] Deng, Z.H., Luo, K.H. and Yu, H.L., 2014. A study of supervised term weighting scheme for sentiment analysis. Expert Systems with Applications, 41(7), pp.3506-3513.

Keywords: natural language processing, document classification, LSA, gradient descent

Supervisor:	RNDr. Kristína Malinovská, PhD.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.
Assigned:	09.04.2018
Approved:	11.04.2018

prof. RNDr. Rastislav Kráľovič, PhD.
Guarantor of Study Programme

....

Student Supervisor

Acknowledgment: I would like to thank my supervisor Kristina Malinovská, for complementing my writing weaknesses. Also, I would like to thank my colleagues from CEAI, where I learned tons of invaluable research and programming skills, without which I would not be able to properly finish this thesis. Special thanks go to Radim Řehůřek, who responded to an email of a random guy (me), who asked him for mentoring and advice. I would also like to express my gratitude to Majka, for valuable suggestions and for her own thesis, which provided me with literary inspirations. Next, I want to thank Guido van Rossum, for creating a tool, which inflicted to me the most pain and caused the most professional joy in my life. Last, but not least, huge thanks goes to Baša, my family, and my friends for supporting me in this funny period of time.

Abstrakt

Latentná sémantická analýza môže zlyhávať pri klasifikačných úlohách, lebo vyberá črty dokumentov, ktoré sú najreprezentatívnejšie, ale nie najdiskriminatívnejšie. V tejto práci predstavujeme novú metódu eLSA, ktorá prináša ďalšiu vrstvu váh w', ktoré sú trénované pomocou metódy najväčšieho vzostupu. Experimentálne sme ukázali, že proces učenia eLSA konverguje, a že eLSA dosahuje väčšiu presnosť ako LSA. Taktiež využívame eLSA na analyzovanie bežne používaných váhových schém a identifikujeme slová, ktoré tieto schémy podhodnocujú alebo nadhodnocujú.

Kľúčové slová: spracovanie prirodzeného jazyka, klasifikácia dokumentov, gradient descent, LSA

Abstract

Latent semantic analysis may perform poorly on document classification tasks, because it selects the most representative, but not the most discriminative features. We propose a new method eLSA, which introduces another layer of weights w' that are trained with gradient descent. We experimentally show, that learning of eLSA converges, and that it achieves higher accuracy than LSA. We also use eLSA to analyze common weighting schemes and identify words, which are underweight or overweight in these schemes.

Keywords: natural language processing, document classification, gradient descent, LSA

Contents

List of Figures

List of Tables

Introduction

The problem of document classification attracted a lot of attention recently. The most popular approaches are based on neural networks and neural word embeddings. We revisit famous co-occurrence-based approach, the Latent Semantic Analysis. This approach may perform poorly on document classification tasks, because it selects the most representative features and not the most discriminative ones. There was a number of attempts to add the supervised information into LSA and into its SVD part. We build on the top of one of these approaches – supervised weighting schemes.

Our goals are to propose a novel approach for learning task-specific word weights, explore performance of this approach and compare it with commonly used weighting schemes.

The main part of this thesis introduces the design of eLSA, approach that employs gradient descent technique to learn task specific supervised weights. Further, we present thorough performance evaluation of eLSA on multiple datasets. Finally, we use eLSA to gain a valuable insight about the classification tasks we want to solve and about commonly used weighting schemes.

In the first chapter, we introduce the problem of document classification and we describe some of the commonly used approaches to this problem. In the second chapter, we present an overview of published literature about introducing supervision into the LSA. In the third chapter, we introduce our novel approach eLSA, which builds on the top of supervised weights and LSA. In the fourth chapter, we present results achieved by the eLSA and possible ways how to extract insight from its parameters.

Chapter 1

Document Classification

In this chapter, we introduce a problem of document classification and commonly used approaches that are used to solve this problem.

Our approach is based on unsupervised learning, which we later enhance by incorporating a few tricks from supervised machine learning.

1.1 Problem description and motivation

Thanks to digitization of books and texts, almost all human knowledge is now easily accessible to algorithms. With rapid growth of social media platforms and electronic communication, we see a new type of documents. These documents are not traditional books, but user generated content. Websites, blogs, Facebook posts, tweets, G+ posts contain vast amounts of information that can be very valuable. Moreover, companies customer centers usually receive large amounts of questions and requests from customers that could be automatically processed.

Automatic text categorization and classification is therefore a very important and fundamental task. A major objective of text classification system is to semantically assign one or more predefined categories to documents.

Is this document a positive review? Is this question about a person? Is this customer ticket related to a software bug? All of the questions above are instances of the text classification problem.

Usually, machine learning, statistical pattern recognition, or neural networks are used to construct classifiers automatically.

1.1.1 Sentiment analysis

One of the most popular instances of document classification is a sentiment analysis – *SA*. SA is a process of determining whether a piece of writing is positive, negative or neutral [27]. It is also known as *opinion mining*, deriving the opinion or attitude of a speaker. A common use case for this technology is to discover how people feel about a particular topic. SA is widely applied to customer materials such as reviews or online and social media. It can answer questions like: “Does this review suggests, that Eat&Meet is a good place to eat (and meet)?”, “Does the person like his dog?”.

There is a lot of ways how to approach document classification problem. The most important things that we consider about those methods are accuracy and interpretability. Further, we consider also the speed of required preprocessing, the speed of actual classification, the amount of required data, and other memory requirements.

In this thesis, we revisit one of the older approaches called LSA and try to address some of its shortcomings.

1.2 Notation

In this section we provide a concise reference describing the notation and terms used in this thesis.

$\circ$	Hadamard product.
α	Learning rate.
d_j	j-th document in our corpus.
D	Vocabulary. Denotes set (dictionary) of all used words in our dataset.
$\lvert D\rvert$	Size of the vocabulary.
E	Loss function such as L2.
N	Number of documents in the corpus.
n_w	Number of times word w appears in the whole corpus.
Q	Cost function, very similar to loss function.
Θ	Model parameters.
$tf_{w,s}$	Number of times word w appears in sentence s.

Our math notation is mostly influenced by programming practice. In this thesis we assume all vectors to be row vectors. We also assume natural broadcasting of some operations, like $\circ$ for the Hadamard product.

Throughout this thesis we try to illustrate things with examples. For consistency, we will use following sample corpus of sentences and labels:

This is an example "pet sentiment dataset". Labels denote, if the sentiment of the statement was positive (1) or not (0).

There are $|D| = 11$ distinct words in the vocabulary D of this corpus:

```
a, bad, cat, charlie, dog, good, is, max, nice, oscar, tiger
```

Later we will refer to them in this ordering, indexing from 1. Hence $D_4 =$ `charlie`.

charlie is a good dog	1
tiger is a bad cat	0
oscar is a nice cat	1
max is a bad dog	0

Table 1.1: Example dataset

1.3 Machine learning

Machine learning explores the study and construction of algorithms that can learn from data and make predictions based on them. More pragmatic description of machine learning is, that it is a magic process of fiddling with some numbers, until results look fine [9]. Being able to let algorithms automatically learn from data proved to be extremely important. Nowadays, almost any task can be improved or automatized thanks to such algorithms. However, to use machine learning for some task, we usually need to specify the task in machine learning friendly way. In this chapter, we introduce some of machine learning concepts in details.

1.3.1 Supervised machine learning

Supervised learning is the most common approach in machine learning. Supervised algorithms learn to predict the best outputs for a given input. We denote the collection of input data, features, as X and the corresponding expected outputs, labels, as Y. X is usually a matrix of real values where rows of this matrix are individual samples. Y is usually a vector of real values or integers. Inside mathematical expression we usually denote labels as y, We refer to the pair of features and labels (X, Y) as a dataset. In practice we usually have three such datasets: train, validation and test. For the sake of this introduction we ignore this fact and we explain it in section 3.3.3.1.

In machine learning, we want to find a function f such that for given sample x_i the functions output $\hat{y_i} = f(x_i)$ is very close to the real label y_i.

This is usually done by optimizing parameters Θ of a parametrized function f_Θ, with regards to a loss function $E_y(\hat{y})$. We refer to function f_θ as a *model*. Common loss is an $L2$ loss function

$$E_y(\hat{y}) = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y_i})^2 = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_\Theta(x))^2$$

Formally we want to find parameter $\hat{\Theta}$ such that $\hat{\Theta} = argmin_\Theta\left(E_y(f_\Theta(x)\right)$. This equation is usually not solved directly, but through an optimization process called learning [19].

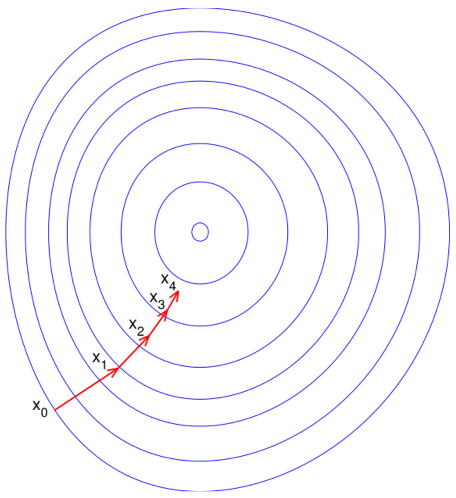

Figure 1.1: Gradient descent [14]

1.3.1.1 Gradient descent

Gradient descent method finds local minima of a usually multivariate function. This is a well suited approach to use in context of supervised machine learning.

We use an observation, that if we follow the opposite direction of the gradient in a given point, we arrive in a local minima. Example is on the picture 1.1.

In the context of machine learning, we optimize a cost function Q of parameters Θ,

$$Q(\Theta) = E_y(f_\Theta(x))$$

We follow the opposite direction of gradient of the cost function Q in respect to parameters Θ. We initialize Θ_0 to small random numbers and perform a gradient descent step

$$\Theta^{t+1} = \Theta^t - \alpha \frac{\partial Q(\Theta^t)}{\partial \Theta^t} = \Theta^t - \alpha \nabla Q(\Theta^t) \tag{1.1}$$

α denotes the size of the step we will make and is commonly known as a learning rate. We perform the gradient descent step until it stops improving the results. Formally we stop, when $|\Theta^{t+1} - \Theta^t| < \epsilon$ for a given ϵ [8].

This process is also sometimes referred to as a batch gradient descent.

1.3.1.2 Stochastic gradient descent

During each gradient descent step, we need to evaluate the gradient of the loss function over the whole dataset. This is usually not feasible for larger datasets.

We can exploit that the cost function Q can usually be rewritten as a sum of costs Q_i for each data point x_i.

$$Q(\Theta) = \sum_{i=1}^{n} Q_i(\Theta)$$

$Q_i(\Theta)$ denotes the cost function computed on only the i-th element. Instead of performing the gradient step on the whole Q, we can perform a gradient step for each Q_i.

$$\Theta^{t+1} = \Theta^t - \alpha \frac{\partial Q_i(\Theta^t)}{\partial \Theta^t} = \Theta^t - \alpha \nabla Q_(\Theta^t) \tag{1.2}$$

For appropriate α we usually see a much faster convergence than for gradient descent.

1.3.2 Feed forward neural network

There is a lot of ways how to construct function f_θ that we want to optimize. One of the most popular ones is roughly inspired by the human brain and is called a *feed forward neural network.*

Neural network consists of small interconnected computational units (neurons) that are usually organized into a layers. Each unit takes some inputs, produces an output based on input and sends it to other units. In a feed forward neural network, the signal is always moving forward, hence unit on the k-th layer can only take its input from previous layers [19].

By adjusting the connections and theirs strengths, the network can learn to produce a specific output for a specific input.

A simple neural network is shown in figure 1.2.

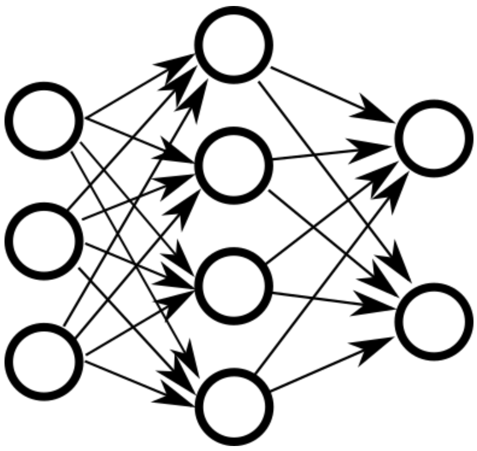

Figure 1.2: Simple neural network with 2 layers, 3 inputs, 4 hidden neurons and 2 outputs

The simplest realization of a unit is a weighted sum and application of activation function g. Formally the unit receives a vector of inputs x and computes output

$g(\sum_{i=1}^{n} \theta_i x_i)$, where θ_i is a connection strength to the i-th input. This one unit can be viewed as a simple one layer neural network with one output.

Common realizations of function g are:

- logistic function $g(x) = \frac{1}{1+e^{-x}}$
- hyperbolic tangent $g(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
- relu $g(x) = \max(0, x)$
- leaky relu $g(x) = \log(1 + \exp(x))$

One layer of such units can be compactly described thanks to matrix notation as

$$y = g(X\Theta)$$

X represents the input matrix (number of samples n times number of features k) and Θ is the matrix of weights (number of features $k \times$ number of units u). Note that function g is applied element-wise.

1.3.2.1 Logistic regression

One of the simplest classifiers and one of the simplest neural networks is the logistic regression. It is an one layer neural network with nonlinearity realized by the logistic (sigmoid) function $g(x) = \frac{1}{1+e^{-x}}$. This classifier has a few nice properties. Derivation of its activation function can be very nicely expressed as $g'(x) = g(x)(1 - g(x))$. Sigmoid outputs values in interval $[0, 1]$ that can be directly viewed as a probability for given class. Because of these properties, the logistic regression is one of the most used classifiers and serves as a benchmark to great number of problems.

1.3.2.2 Multilayer neural networks

The important observation about neural networks is, that we can chain such layers to form a deeper neural network.

$$y = g_2(g_1(X\Theta_1)\Theta_2)$$

W_1 are weight of hidden neurons (first layer) and W_2 are weight of output neurons (second layer). Note that g_2 (usually sigmoid) can be a different function than g_1 (usually relu). There is no consensus in the community about how to count the layers. For example the simple network on picture 1.2 could be seen as a 3 layer neural network, because it has input units, hidden units and output units. In this thesis we will use the number of matrices W that are in the model.

In general, such multilayer feed forward neural networks represent very broad family of functions. They are in fact an universal approximator and can approximate any other function [16]. However, we need to train them first.

1.3.2.3 Backpropagation

Backpropagation is a method for efficient computation of the gradient of a neural network. It relies on the fact, that neural networks are just simple compositions of matrix multiplications and function applications. It also relies on the fact, that functions used in the units are differentiable (almost everywhere).

Backpropagation works in two steps, forward pass and backward pass. In the forward pass we feed input to the network, compute activations of each layer and compute the final output. In the second pass, we use the chain rule and incrementally compute gradient with respect to each layer of the network.

Finally we use the gradient descent according to equation 1.1 and optimize the parameters [55].

1.4 Word and sentence representation

In order to apply machine learning techniques to a text classification, we need to represent the text in a machine learning friendly way. Most machine learning algorithms expect the input in a form of vectors of numbers.

Because of this, we need a system to translate string sentences s into vectors of numbers e. Such vector is usually called an embedding.

In other words, we need to project words, sentences and documents into a vector space. We consider the sentences to be basically identical to documents as they both can be considered to be sequences of words.

1.4.1 Local representation

The simplest vector space is based on a local representation. In this representation, we use a vector space with $|D|$ dimensions, one for each word in the vocabulary. Value v_i at the i-th position of the vector v corresponds to a presence or an absence of the i-th word D_i from the vocabulary D. Note that $|D|$ in real applications easily exceeds 100 000 or even 1 000 000. This may make this representation unusable for some applications (classification with SVM).

One of the simplest local representations is a *bag of words* (BOW) representation. In this representation, values in vectors are binary, where 1 means that the word was present in the text and 0 means it was not. This can also be viewed as a simple one hot encoding. Optionally we can extend this to a term vector, where the number is not binary, but it expresses the real count of given word in the sentence.

Looking at our example corpus, our vector space would have 11 dimensions and the first sentence would be represented as a vector

$$e = (1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0)$$

.

This representation allows for simple sentence comparison. Sentences s_1 and s_2 can be compared by comparing cosine similarity of their BOW embeddings e_1 and e_2.

$$\text{sim}(s_1, s_2) = \text{sim}(e_1, e_2) = \frac{e_1 e_2}{||e_1||.||e_2||}$$

For technical reasons we can also define a cosine distance.

$$\text{dist}(s_1, s_2) = \text{dist}(e_1, e_2) = 1 - \text{sim}(e_1, e_2) = 1 - \frac{e_1 e_2}{||e_1||.||e_2||}$$

However, this representation does not consider, that two words can be similar, even though they are not the same. For example words `nice` and `good` are considered to be completely different, even though they have similar meaning.

Second problem is, that this representation forgets the initial ordering of the words. We can address this problem by using pairs of words instead of single words. Pairs of words are called bigrams. For example, the first sentence in our example dataset can be represented as bag of bigrams:

$$e = ((4, 7), (7, 1), (1, 6), (6, 5))$$

Third problem is, that this representation assigns the same weight to each word. The fact that two sentences both have a meaningless word `a` has the same effect on the similarity as if they both have semantically meaningful word `bad`. This problem is partially solved by introducing term weights.

1.4.1.1 Term-weighting schemes

To emphasize that some words in the corpus are more important than others we introduce a term-weighting schemes. The idea is, that words like prepositions (`a`, `an`) and other words that are not very informative or redundant (`is`, `do`, `by`) should have smaller weights. In practice, a list of *stop words* is used to completely filter such words.

Also words, that are to common in the dataset should have lower weights. If each sample is about a dog, we probably do not care about the word `dog`, because it is somehow redundant.

However words, that appear multiple times in a sentence are probably important for this sentence and should have higher weights.

Based on these observations, a number of term-weighting schemes was proposed [56]. A term-weighting scheme assigns a weight to a word w in a sentence s as term weight

$t_{w,s}$. These weights usually consist of two parts, *term frequency* and *inverse document frequency.*

Term frequency part $\mathrm{TF}_{w,s}$ reflects how important is a given word w for sentence s. Common choices are:

- $\mathrm{sign}(\mathrm{tf}_{w,s})$, binary presence of the word in sentence. Also known as BOW.
- $\mathrm{tf}_{w,s}$, number of appearances.
- $\frac{\mathrm{tf}_{w,s}}{|s|}$, number of appearances normalized by length of the sentence.
- $1 + \log(\mathrm{tf}_{w,s})$.

Inverse document frequency part $\mathrm{IDF}_{\mathrm{w}}$, reflects how important is the word w for the whole corpus. Common choices are:

- 1, unary.
- $\log\left(\frac{N}{n_w}\right)$, inverse document frequency.
- $\log\left(1 + \frac{N}{n_w}\right)$, smoothed inverse document frequency.
- $\log\left(\frac{max_{w'} n_{w'}}{n_w}\right)$, max inverse document frequency.
- $\log\left(\frac{N - n_w}{n_w}\right)$, probabilistic inverse document frequency.

Combination of such parts is called TF-IDF. Note that these choices for $\mathrm{TF}_{w,s}$ and $\mathrm{IDF}_{w,s}$ are probabilistically grounded [2].

Another popular weighting scheme that is usually used in document retrieval is BM_{25}.

$$\mathrm{BM}_{25\ w,s} = \log\left(\frac{N - n_w + 0.5}{n_w + 0.5}\right) \frac{\mathrm{tf}_{w,s}(1.2 + 1)}{\mathrm{tf}_{w,s} + 1.2\left(1 - 0.75 + 0.75 \cdot \frac{N}{\mathrm{avgdl}}\right)}$$

This representation performs better than BOW and usually works the best for large documents in document retrieval [36].

Term weights can help to describe that different words have different importance. However, they still cannot take into account, that two different words should have similar representation.

1.4.1.2 Supervised weighting schemes

Number of researchers recognized supervised weighting schemes as an elegant form of feature engineering for document classification problem. We try to build upon these results and upon interesting schemes they proposed.

Our work is mainly based on schemes described by Wu et al.[61], Lan et al. [29], and Deng et al. [18].

We will use schemes `tfchi2`, `tfig`, `tfgr`, `tfor`, and `tfrf`. The exact formulas for these weights are not relevant to our work and we refer the reader to cited articles. The important part is, that these weighs consider how many times the given word appeared in each category as a contrast to unsupervised weights.

1.4.2 Distributed representation

To address the problem of high dimensionality and to allow different words to have high cosine similarity, we employ distributed representation. In this representation, dimensions do not correspond directly to words, but rather to some features of these words. In this representation the ”meaning“ of a word is split across multiple different dimensions. We can view each dimension as if it holds some form of an abstract meaning [31].

Simple example of distributed representation is a binary representation of a number.

To acquire a distributed representation, algorithms commonly make use of the distributional hypothesis.

1.4.2.1 Distributional hypothesis

Distributional hypothesis states, that words which appear in similar contexts tend to have similar meaning, even though they do not appear directly together [21] [53]. For example we can exchange words `dog` and `cat` in each sentence in our example corpus 1.1 and all sentences will still make sense. The positive correlation between the words appearing in similar contexts and words having similar meaning was in fact empirically confirmed.

Because of this, distributional hypothesis is used as a foundation for a lot of methods that try to capture the word semantic similarity [54].

1.4.3 Prediction based distributed representation

Number of researchers used neural networks to create a good, dense, distributed word representations [48] [32] [52]. These representations are also commonly called *neural embeddings* or just word vectors. We will call these representations *word vectors.* Word vectors proved to be very useful in broad range of natural language processing tasks.

Moreover, they manifest interesting algebraic properties.

$$v(king) - v(man) + v(woman) = v(queen)$$

$$v(better) - v(good) + v(small) = v(smaller)$$

v denotes the word vector for given word.

There are two main approaches for training word vectors with neural networks: skip-gram model and continuous bag of words model. Based on an embedding of an input word x, the *skip-gram model* predicts words from the context of x . On the other hand, the *continuous bag of words model* tries to predict the word based on its context.

These methods can be computationally intensive, and they require a lot of training data. Moreover their training usually requires a lot of well chosen hyper-parameters and tricks [41] [58].

1.4.4 Count based distributed representation

Count based methods create co-occurrence matrix and try to extract its underlying structure. These approaches are very popular before the raise of neural networks and neural embeddings.

There is multiple ways how to extract some structure out of the Co-occurrence matrix. For example, latent Dirichlet allocation (LDA) [7] constructs a probabilistic model of each document. It assumes, that each document is created as a mixture of topics and that each topic is just a distribution over words.

Second popular class of approaches relies on performing some matrix factorization on the co-occurrence matrix. In this thesis we build on one of these factorization approaches.

1.4.4.1 Latent semantic analysis

For the purposes of this thesis we consider the *latent semantic indexing* (LSI) to be the same as *latent semantic analysis* (LSA) [17]. For simplicity, we will refer only to the LSA.

Latent semantic analysis is one of the standard approaches to identify hidden variables describing the data. In the context of natural language classification it extracts and infers relations of expected contextual usage of words in documents. It does so without any humanly constructed dictionaries, knowledge bases, semantic networks, grammars or syntactic parsers.

This property proves to be extremely useful and is not specific only to the LSA. Machine learning algorithms that rely only on data X and do not need any form of labels Y are commonly called unsupervised. The problem is, that data is usually very easy to acquire (just set of some documents), but labels Y are very hard and costly, because they may require human annotation.

LSA starts with a co-occurrence matrix M'. Each cell $M'_{i,j}$ of this matrix contains number representing the frequency with which the word D_i appears in the document

d_j. Afterwards each cell entry is weighted by a function that expresses both the words importance in the particular document and the word importance in the general corpus. A weighting scheme such as TF-IDF can be used. We denote the reweighted matrix M.

Next, LSA applies singular value decomposition (SVD) to the matrix. In principle, SVD achieves a two-mode factor analysis and positions both terms and documents in a single space defined over the extracted dimensions. In case a different matrix factorization is used such as non negative matrix factorization, we would arrive to probabilistic latent semantic analysis.

SVD breaks down the document matrix M into three matrices U, Σ, V,[46] [37] [60] [30] such that

$$M = U\Sigma V^T$$

U and V are orthogonal matrices and Σ is a diagonal matrix.

$$\begin{array}{ccccccc} M & & U & & \Sigma & & V^T \\ \mathbf{d}_i^T & & & & & & \\ \downarrow & & & & & & \\ \begin{bmatrix} x_{1,1} \dots x_{1,N} \\ \vdots \quad \ddots \quad \vdots \\ x_{j,1} \dots x_{j,N} \\ \vdots \quad \ddots \quad \vdots \\ x_{|D|,1} \dots x_{|D|,N} \end{bmatrix} & = \quad \mathbf{u}_j \rightarrow & \begin{bmatrix} [\ \mathbf{u}_1\ ] \\ \vdots \\ [\ \mathbf{u}_{|D|}\ ] \end{bmatrix} & \cdot & \begin{bmatrix} \sigma_1 \dots & 0 \\ \vdots & \ddots \vdots \\ 0 \dots \sigma_k \end{bmatrix} & \cdot & \begin{bmatrix} \begin{bmatrix} \\ \mathbf{v}_1^T \\ \\ \end{bmatrix} \dots \begin{bmatrix} \\ \mathbf{v}_N^T \\ \\ \end{bmatrix} \end{bmatrix} \end{array}$$

U describes the original row entities (words) as vectors of derived orthogonal factor values. V describes the original column entities (documents) in the same way. Σ contains scaling values such that when the three components are matrix multiplied, the original matrix is reconstructed.

It was shown that any matrix can be decomposed perfectly in a such way, using no more factors than the smallest dimension of the original matrix.

Moreover we can construct an approximation with reduced rank by setting all but the k biggest entries in Σ to zero. This effectively reduces the number of columns of U to k and similarly number of rows of V^T. The thesis behind LSI is that those less important dimensions correspond to "noise" due to word-choice variability.

Note that due to this reduction we inevitably loose some of the information. We can see this as forgetting some of the less often words.

An interesting property of SVD is that the generated approximation is the closest matrix of its rank to the original in the least-squares sense [5].

In practice, computing the full SVD decomposition would be time and memory demanding and even unfeasible. Because of that, we directly compute the lower rank approximation with k dimensions, such that $M \simeq U_k \Sigma V_k^T$ [20]. Moreover, we do not even need to have access to the full matrix M and we can compute the decomposition in an incremental manner [10].

We can incrementally adjust the decomposition when adding a new document into the corpus, moreover we can remove a document as well. Incremental SVD could in theory be used in the stochastic gradient descent setting.

From now on, we drop the subscript k and matrices U, Σ and V are assumed to have only k dimensions in the later text.

1.4.4.2 LSA for document classification

In the context of document classification, we first construct our vocabulary D from our training data. We can filter out words that we know are not relevant, or the ones that appeared only few times in the corpus.

Now we can transform sentences into their BOW representations. Afterwards, we compute all necessary statistics required by our chosen weighting scheme. In case of TF-IDF, we compute the number of appearances of each word in our text n_w. This can be done efficiently by making use of fast matrix operations and libraries implementing linear algebra. Having these statistics, we can reweigh all words with them and construct the term matrix M.

Then we factorize M an compute the matrices U, Σ, V. In theory, we could directly read the document embeddings v from the matrix V. For practical purposes, the matrix V_k is usually not stored and sometimes not even computed.

To compute the embedding, we take the document term vector d_i and employ a relationship

$$d_i^T = U \Sigma v_i^T$$

Because U is orthogonal and Σ is diagonal, we get

$$\Sigma^{-1} U^T d_i^T = v_i^T$$

$$v_i = d_i U \Sigma^{-1}$$

Note that Σ^{-1} is easy to compute, because Σ is a diagonal. We just need to invert each number on the diagonal.

It was experimentally shown in my bachelor thesis [38] and by other researchers [35] that Σ may not reflect the importance of each feature very well. Because of that, a simpler relationship is usually employed

$$v_i = d_i U$$

This process can have a different interpretation. Each entry in matrix U is the distributed representation for one word. The matrix multiplication selects words, that are present in the document and computes a weighted sum of them.

Important part is, that we can compute such embedding even for unseen documents. The problem is, that we have to filter all words that are not present in D. However, because this method is fully unsupervised, we can make use of any unlabeled data. Factorized matrix M can contain documents, that we do not have the label for.

Finally, we perform the classification. Matrix of embeddings v_i are considered to be the document features and are used as an input X to some classifier. A popular choices for classifier are a SVM or neural networks. First we train the classifier on some training set of documents and ther labels. Then we can easily predict labels for unseen documents.

We can break down the whole process into three parts: weighting, decomposition and classification. We denote the weighting part as W, the decomposition part as S (SVD) and the classification part as C.

1.4.4.3 LSA pitfalls

Even though LSA was shown to perform well in document retrieval applications and it can capture some basic semantic properties of words, it has certain limitations when applied to classification tasks.

The problem is, that LSA does not incorporate the information about document classes. It finds the most representative features and not the most discriminative ones [5]. In other words, it finds embedding that is the best for reconstructing the documents. Because of that, infrequent words that have high discriminatory power (are important for the task), may be filtered out.

For example, LSA computed on our example corpus 1.1 would focus on capturing the words `a`, `is`, `dog` and `cat`, event though they are completely irrelevant to our prediction task.

1.4.5 Prediction vs. count based representations

LSA and other count based techniques used to be very popular alternative to local representations. However, after the raise of neural networks, they were soon overshadowed by neural word embeddings.

Prediction based models achieved much better performance across multiple word relatedness tasks, categorization tasks, synonym tasks and analogy tasks [4].

During these evaluations, they obtained the word representations by training on a big monolingual dataset, like wikipedia. Afterwards they used prepared pairs of related words (synonyms or words from the same category) and tested how similar are the vector representations of related words. They measured how well the word vector similarity correlates with the actual relatedness. They report that count based methods achieved Pearson correlation coefficient 0.74 and were outperformed by prediction based methods which achieved 0.84 [4].

This and the hype around neural networks led to recent extreme popularity of prediction based embeddings, even though they were much less understood and less ”mathematically grounded“.

Later it was shown, that these two approaches are in fact very similar. It was proven, that a prediction based model skip-gram with negative-sampling (SGNS) is implicitly factorizing a word-context matrix M, whose cells are the point-wise mutual information (PMI) of the respective word and context pairs, shifted by a global constant [34].

These results suggest, that these two approaches are somehow equivalent and with proper weights the LSA should achieve similar results as skip-gram model. However, this was not the case in practice.

Prediction based models require a lot of hyperparameters, that cannot be learned and need to be set up manually, or through excessive search. On the other hand, count based models usually do not require a lot of parameters, besides the number of dimensions of the resulting embedding.

Levy et al. [35] showed, that much of the performance gains of neural embeddings are due to such hyperparameters and certain system design choices, rather than the embedding algorithms themselves. Furthermore they showed that traditional (count based) distributional models can also benefit from such hyperparameters and design choices.

After incorporating these insights into the count based methods, they observed mostly local or insignificant performance differences between quality of count based embeddings and prediction based embeddings.

Note that these experiments were done on a large bodies of text like English Wikipedia. On small datasets it was empirically shown, that count based methods actually outperform the neural networks[3].

Chapter 2

Similar work

From studying the literature, we identified 3 ways how to address shortcomings of LSA for document classification. In this chapter we discuss these approaches in more details.

We can modify the term matrix M so that weight of each word represents how important is this word for the actual classification task and use *supervised term weights*. This is a contrast to the classical approach, where the weight of the word represents how important is it for the document reconstruction.

Besides that, we can introduce some form of supervision directly into the matrix decomposition process. Sun et al. [57] propose supervised latent semantic indexing (sLSI) and they employ the supervision while choosing the most relevant dimensions of the embedding. Chakraborti et al. [12] propose to add class dependent terms (supervised terms) into the term matrix and proceed with standard factorization.

Finally, we can fully employ advances made in artificial neural networks and use LSA just as a module into these networks.

In this chapter, we introduce each of these approaches in more details.

2.1 LSA with class knowledge

2.1.1 Supervised LSI

Recall (from section 1.4.4.1) that we consider LSI and LSA to refer to the same method.

Sun et al. [57] also recognized that LSI is not optimal for document categorization tasks because it finds the most representative features and not the most discriminative ones. To address this problem, they incorporate the available information about the classification of documents into c categories and propose supervised latent semantic indexing (sLSI).

To introduce their approach we will recapitulate the classical LSI first. LSI selects the eigenvector of the term matrix M with the biggest eigenvalues. The effect of this basis vector is then removed from the original document vector and we iterate this process.

SLSI algorithm employs the class information to guide the selection of the basis vectors. For each category they perform standard SVD on the matrix M. Then they collect eigenvectors with the highest eigenvalues into a collection of candidate vectors.

For each of these candidate vectors g_s they project all documents onto this vector. They compute centroids (average position of documents from given class) of each class

$$v_s^p = \frac{1}{|C_p|} \sum_{d \in C_p} g_s d^T \tag{2.1}$$

where C_p denotes all documents belonging to class p. Note that because they project onto one vector, this centroid is a scalar. Then they compute distances between such centroids for each pair of classes $f_s^{pq} = |v_s^p - v_s^q|$.

Finally, they select the most discriminative vector as vector that separates some pair of classes the best

$$b = arg_{g_i \in G}\{\max_{pq} f_i^{pq}\} \tag{2.2}$$

and proceed to select the next one.

They report that this algorithm generates better representations than standard LSI and can achieve higher dimensionality reduction without reducing the classification accuracy.

2.1.2 Sprinkling

Chakraborti et al. propose another way how to incorporate class information into LSI [12][13]. They propose to add the label directly to the term matrix M resulting into a "sprinkled" matrix M'.

Label related terms $y_{i,k}$, the sprinkled terms, have value 1 if the document d_i belongs to the class k and 0 otherwise. These terms are added as normal words into each document. Note that Chakraborti et al. use transposed form of M (documents are rows of M). We decided to use this variant to be consistent with the rest of the thesis (section 3.2.

After obtaining the matrix M', they perform standard SVD. Then they "unsprinkle" the matrix by removing word vectors associated with sprinkled terms.

This approach reports 1% improvement over LSI without sprinkling.

$$\begin{array}{c} M \\ \mathbf{d}_i^T \\ \downarrow \\ \left[\begin{array}{ccc} x_{1,1} & \cdots & x_{1,N} \\ \vdots & \ddots & \vdots \\ x_{j,1} & \cdots & x_{j,N} \\ \vdots & \ddots & \vdots \\ x_{|D|,1} & \cdots & x_{|D|,N} \end{array}\right] \end{array} \qquad \begin{array}{c} M' \\ \mathbf{d}_i^T \\ \downarrow \\ \left[\begin{array}{ccc} x_{1,1} & \cdots & x_{1,N} \\ \vdots & \ddots & \vdots \\ x_{j,1} & \cdots & x_{j,N} \\ \vdots & \ddots & \vdots \\ x_{|D|,1} & \cdots & x_{|D|,N} \\ \hline y_{1,1} & \cdots & y_{N,1} \\ \vdots & \ddots & \vdots \\ y_{1,k} & \cdots & y_{N,k} \end{array}\right] \end{array}$$

Figure 2.1: Effect of sprinkling on term matrix

2.1.2.1 Why does it work?

Work of Kontostathis et al. [28] reveals close correspondence between LSI and higher order associations between terms. We say that word w_1 has a first order association with another word w_2, if they co-occur in at least one document. We say that words w_1 and w_2 share a second order association if there is at least one term w_3 that co-occurs with w_1 and w_2 in distinct documents. Kontostathis et al. provide experimental evidence to show that LSI boosts similarity between terms sharing higher order associations. Sprinkled terms create such higher order associations between terms related to the same class/category.

2.2 Supervised weights

Term weighting is a strategy that tries to improve the performance of sentiment analysis or text mining tasks by assigning weights to words. If the reweighted vectors are used for some form of classification, we can enhance the weighting scheme by the class information.

There is a number of ways, how to design such supervised term weights. In this thesis, we will build on upon of some of those schemes, because we believe that supervised weighting schemes are a good approach to combat the shortcomings of LSA for document classification.

In this chapter we introduce an interesting use of LSA with supervised weights for the problem of semantic relatedness.

2.2.1 TF-KLD

Ji and Eisenstein [26] studied the performance of LSA for semantic relatedness task. In semantic relatedness task we have two documents d_i^1, d_i^2 and we want to determine, whether they are semantically similar or not. Example of semantic relatedness task can be to determine, if two questions asked on some forum are the same (and should be deduplicated) or not [1].

For this task, Ji and Eisenstein proposed to compute the LSA embeddings of both texts v_i^1 and v_i^2 and train classifier on features based on these embeddings. In the paper they used $[v_1 + v_2, \|v_1 - v_2\|]$.

They recognized, that LSA with term matrix M, containing term frequencies, performs poorly. They designed novel supervised weighting scheme called TF-KLD (term frequency, Kullback-Leibler divergence). This weighting scheme has classical term frequency part and second novel KLD part.

KLD part describes, how important is given word for the task. They compute probabilities

$$\begin{aligned} p_j &= P(s_{ij}^1 | s_{ij}^2, y_i = 1) \\ q_j &= P(s_{ij}^1 | s_{ij}^2, y_i = 0) \end{aligned} \tag{2.3}$$

In other words, what is the probability that word w_j is in the first sentence if it is the second one and the sentences are (not) related.

KLD for word j is

$$KL(p_j || q_j) = \sum_x p_j(x) \log \frac{p_j(x)}{q_j(x)} \tag{2.4}$$

, where KL denotes the divergence. This can be seen as an amount of information that presence of the word tells us about similarity of this pair of sentences.

This approach with TF-KLD weights can perform even better than recurrent neural networks [15]. This result shows, that with proper supervised weights, we can even outperform complicated neural network approaches.

2.3 SVD in neural networks

Ionescu et al. [25] proposed a very interesting method of how to use good properties of SVD inside a neural network. They researched the field of computer vision, that is currently dominated by convolutional neural networks (CNNs) [33].

They argue, that CNNs specialize at local computations while on the other hand SVD can perform global computations.

[1] `https://www.kaggle.com/c/quora-question-pairs`

They propose to use matrix factorization as a module inside the neural network. They also proposed a *matrix backpropagation* methodology to train these networks. They compare their approach against widely used convolutional architectures VGG and Alexnet and report that addition of such structured layer significantly improved the segmentation accuracy.

They compute the exact update that is necessary to perform on matrices U, σ, V that describe the SVD. However, this computation is only possible, because they use the full SVD (not only the top k dimensions). Because of this we cannot directly use their approach. However, their results show, that it is possible to "flow the gradient through SVD". Hence gradient descent can be used to update parameters that are used before the SVD.

Chapter 3

eLSA

In this chapter we introduce our novel approach eLSA: error boosted LSA. We describe our motivations, the design of eLSA and some implementation details.

3.1 Goals and motivation

The root of LSA's problems is the dimensionality reduction. It will inevitably throw away some information. However, this information may be vital to the classification task. It is clear, that we need to address this problem and introduce some form of supervision, that precedes the SVD.

To some extend, this can be done by the supervised word weights described in section 1.4.1.2. However, we consider this supervision to be rather weak, as it only computes simple statistics over the words and labels.

In the end, we want to train some classifier C on top of the document embeddings. We would like to find a way, how to use the knowledge about the classifiers error and incorporate it before the SVD. We achieve it by training our own word weighting scheme.

We propose a novel approach for learning task-specific word weights. We experimentally evaluate the performance of this approach and compare it with other (commonly used) weighting schemes. Finally we use its properties to acquire new insights about the dataset.

As a side goal, we would like to introduce as few new hyperparameters as possible, because they may cause a "false improvement" of our approach over the baseline. We also would like our approach to be at least slightly interpretable. It would be nice, if it could produce some insight to the dataset.

Because of the supervised weights, we know that supervision helps. The only problem is, how to achieve it. Here we take the inspiration from neural networks, and the insight that we can easily optimize parameters of rather complex system, as long as each part of this system is differentiable.

3.2 Design overview

First, recall how does the LSA works in the context of document classification. We start with a sentence s_i and we represent it as a bag of words vector d'_i. Then we transform it with a weighting scheme into a term vector d_i. Afterwards we compute the lower dimensional embedding of the document $v_i = d_i U$. Finally we use this embedding v_i as an input x_i to the classifier C (logistic regression, neural network or SVM). This classifier predicts label $\hat{y}_i$ that can be compared to the true labels y_i.

Our contribution is to introduce another layer of reweighting. Instead of using the term vector d_i, we introduce weights w' and use reweighted vector $d_i \circ w'$[1]. This is equivalent to multiplying the vector d_i by diagonal matrix with diagonal w'. Then we perform the SVD decomposition on the reweighted matrix $M \circ w'$. Here, we employ a few conventions from popular linear algebra library NumPy [43], where such operation is automatically broadcasted through the matrix.

We initialize the w' to be a vector of ones (this effectively does not change the matrix at first). For fixed decomposition $M \circ w' = U\Sigma V^T$ and some classifier C (like a neural network), we compute $\hat{y} = C(M \circ w'U)$. Then we want to optimize w' to minimize the error function $E_y(\hat{y})$. The only question is, how to do it.

The natural choice is to employ the gradient descent algorithm. However to use it, we need to be able to compute the gradient of error with regards to the weights $\frac{\partial E_y(\hat{y})}{\partial w'}$.

3.2.1 Gradient computation

To compute the gradient efficiently, we use the backpropagation method mentioned in section 1.3.2.3. We can compute the derivation $\frac{\partial E_y(\hat{y})}{\partial w'}$ thanks to the chain rule

$$\frac{\partial E_y(\hat{y})}{\partial w'} = \frac{\partial E_y(\hat{y})}{\partial \hat{y}} \frac{\partial c(x)}{\partial x} \frac{\partial x}{\partial w'} \tag{3.1}$$

The first part of expression 3.1 is dependent on the chosen error function. For L2 error we get

$$\frac{\partial E_y(\hat{y})}{\partial \hat{y}} = \frac{\partial \frac{1}{2}(\hat{y} - y)^2}{\partial \hat{y}} = \hat{y} - y \tag{3.2}$$

The second part is dependent on the classifier that we use. In case we use logistic regression $\hat{y} = \sigma(x\Theta + b)$, the expression becomes

$$\frac{\partial c(x)}{\partial x} = \hat{y}(1 - \hat{y})\Theta \tag{3.3}$$

[1]Recall, that $\circ$ means Hadamard product (pointwise multiplication).

Note that it is possible to compute this expression even for more complicated classifiers. Some libraries can even compute this automatically [1]. However we need the classifier to be differentiable so we cannot easily use SVM as the classifier.

The third and last part is tied to the computation of the LSA embedding. Because $x = d \circ w'U$, we can compute

$$\frac{\partial x_j}{\partial' w_i} = d_i U_{i,j} \tag{3.4}$$

This can be rewritten in more concise notation as

$$\frac{\partial x}{\partial' w} = U \circ d^T \tag{3.5}$$

We again employ the convention that the $\circ$ operation is broadcasted across all columns of the matrix.

Once we know how to compute the gradient, we need to decide on what optimization routine we will employ.

3.3 Optimization routines

Inspired by gradient descent algorithm, we perform the gradient step

$$w' = w' - \beta \frac{\partial E_y(\hat{y})}{\partial w'} \tag{3.6}$$

The parameter β determines the size of the gradient step that we will perform. We will call this parameter w'-learning rate. Note that the classifier C may have its of learning rate α, and these two learning rates are completely independent.

The problem is, that equation 3.1 holds only for fixed classifier C and for fixed decomposition S. We ignore the fact, that S is actually a function of reweighted matrix and hence it is a function of w'. It is probably possible to compute the exact gradient (refer to section 2.3), but we will not do that.

It is not clear, how often we want to recompute the SVD and how often (and how much) we want to retrain the classifier. We propose two main approaches inspired by neural network training and by expectation minimization algorithm.

We use the notation introduced in section 1.4.4.1. However, our parts W, S and C are going to be updated through the time. We denote them with superscripts, e.g. W^t, referring to weighting part W at time t.

3.3.1 Batch gradient descent

We need to find weights w' for the part W, such that the decomposition S captures the most information and hence the classifier C has access to all the important features of the document. However, the decomposition S is dependent on the reweighted matrix produced by W. Because of this, after each update of w' we need to fully recompute the decomposition S and retrain the classifier C. The first optimization routine is described in the algorithm 1.

```
**Data:** $M$
**Result:** trained $W$, $S$, $C$
$w'^0 = 1$, $t = 1$;
**while** *performance is improving on validation dataset,* $t++$ **do**
    recompute $S^t$ from $M \circ w'^{t-1}$;
    compute embeddings $v^t$ from $M \circ w'^{t-1}$ with $S^t$;
    fully train $C^t$ on $v^t$ and labels $y$;
    update $W^t$: $w'^t = w'^{t-1} - \beta \frac{\partial E(C(S(Mw'^{t-1}))}{\partial w'^{t-1}}$;
**end**
```

Algorithm 1: stochastic training of w'

We took inspiration from expectation maximization algorithm. Note that it is not essential in any way for the reader to know this algorithm for further reading.

We name the algorithm 1 eLSA, because it uses error of the classifier to produce boosted (improved) weight w'.

This algorithm needs to do a lot of computations for one update of the weights w'. To address this, we can increase the number of gradient steps performed on w' in each step of the while loop. We recall the same problem in neural networks was solved by stochastic gradient descent. We explore such solution as well.

3.3.2 Stochastic gradient descent

We take the inspiration from stochastic gradient descent step from neural networks. Instead of computing the full gradient, we just compute the gradient related to one example

$$\frac{\partial E_{y_i}(\hat{y}_i)}{\partial w'}$$

Because the document vector d_i is sparse, we will only update a few values from w', hence we can easily update part W.

Because we updated w', the entry in matrix M corresponding to the document d_i has changed. We know, that SVD can be build incrementally and that we can add new documents to the matrix M [10]. We just use the adding routine and add the new representation $d_i \circ w'^t$ to the decomposition. Because we changed the reweighted document and the decomposition matrices, the embedding of this document should also change to

$$x_i^t = v_i^t = d_i w'^t U^t \tag{3.7}$$

We update C in the similar manner. We perform the stochastic gradient step for example x_i^{t+1} and threat it as a new example in the dataset. Pseudocode is available in algorithm 2.

```
Data: $M$
Result: trained $W$, $S$, $C$
$w'^0 = 1$, $t = 1$;
compute $S^0$ on $M \circ w'^0$;
compute embeddings $v^0$ from $M \circ w'^0$ with $S^0$;
fully train $C^0$ on $v^0$ and labels $y$;
while performance is improving on validation dataset do
    for document $d_i \in M$, $i++$, $t++$ do
        add $d_i$ to $S^{t-1}$ and get $S^t$;
        compute embedding $v_i^t$ from $d_i \circ w'^{t-1}$ with $S^t$;
        perform gradient step on $C^{t-1}$ with $v_i^t$ to get $C^t$;
        update $W^t$, $w'^t = w'^{t-1} - \beta \frac{\partial E(C(S(d_i \circ w'^{t-1}))}{\partial w'^{t-1}}$;
    end
end
```

Algorithm 2: Batch training of w'

We done a few preliminary experiments and this optimization routine showed to be very unstable. It also introduced a lot of new decisions that would have had been made and a lot of new hyperparameters. We do not explore it further.

3.3.3 eLSA explanation

We need to address a few things that are no apparent from the description provided above. We do it here to not clutter the notation and so that the previous sections are a bit cleaner.

3.3.3.1 Train test split

Motivated by good practice from neural network training, we split our dataset into 3 parts: train, validation, and test. Train set is 80% of our samples, validation and test are both 10%. We make sure that all those sets have percentage of positive and negative labels that is similar to the whole dataset.

We use the train dataset for gradient evaluations and overall training. We use the validation dataset to keep track of our performance on unseen data. Based on this performance, we decide if the model is still improving, or if it started to overfit. Finally we report the performance on the test set, that was never seen by the model.

Because our datasets are not very big, we stabilize our results by employing the crossvalidation technique. However because we need to run a lot of experiments, we only use 3 fold crossvalidation.

3.3.3.2 Stopping conditions

In both optimization routines we iteratively update parameters w' and we do so while our performance on validation set is increasing. This needs more clarification.

In each time step t we evaluate the loss E^t_{valid} on the validation dataset. By convergence we mean, that the E_{valid} stopped improving. In the final solution we track the mean of this errors in last few time steps. When this mean does not increase for long enough, we stop the training. Formally we track $ME^t_{10} = \frac{1}{10}\sum_{0\leq i<10} E^{t-i}_{valid}$ (mean for last 10 iterations) and $ME^t_{10,20} = \frac{1}{10}\sum_{10\leq i<20} E^{t-i}_{valid}$ (mean for previous 10 iterations).

We stop the training in time t if $ME^t_{10} > ME^t_{10,20}$, which means our validation error started to increase and we need to stop.

3.3.3.3 Weighting schemes

An important properly of eLSA algorithm is, that we do not need to start from basic co-occurrence matrix M. We can apply some weighted scheme, such as TF-IDF on this matrix first and then run eLSA. What weighting scheme we starts from is actually one of the important hyperparameters.

Afterwards, we can examine the found w' and see, which words had high value w'_i. This means, they were underweighted by the weighting scheme.

3.3.3.4 Parameters

eLSA on its own has only three hyperparameters and that is the w'-learning rate β and the initial weighting scheme. Second hyperparameter is the dimensionality of the embedding. This hyperparameter is not specific to eLSA. However eLSA can be viewed as a tool for analyzing weighting schemes so it is actually no a hyperparameter.

In fact the SVD decomposition (part S) and the classifier (C) may have a lot of hyperparameters on their own. Exploring the space of those hyperparameters would be extremely computationally demanding and could be subject to further research. Instead we leave this parameters to reasonable defaults suggested by implementations we use. We can do this, because we are not interested in pushing the state of the art in document classification problem, we just want to see a relative improvement over the LSA baseline.

3.4 Implementation

In this section we go through some implementation details and decisions. The whole thesis, including experiment evaluations, is implemented in Python3. For code, please refer to appendix A

3.4.1 Libraries

While implementing our approach, we made use of multiple tools and libraries that deserves to be mentioned.

3.4.1.1 Scikit-learn

Scikit-learn (or sklearn) is a free machine learning library for Python programming language [47]. It features wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. It is extensible and provides very nice and clean APIs [11].

In this thesis, we used it for multiple purposes. We used base classes for weighting schemes `TfidfTransformer`, `BaseEstimator`, `TransformerMixin`. We also use this library to generate training and test splits.

Lastly, we used scikits implementation of `LogisticRegression`. However, in our approach we need to compute a derivative of our classifiers. This implementation does not provide an API for this computation and we had to use its internal variables `coef_` and `intercept_` that corresponds to the weights and bias of the classifier.

3.4.1.2 Gensim

Gensim is a robust open-source vector space modeling and topic modeling toolkit implemented in Python [50]. It is an efficient and hassle-free software to realize unsupervised semantic modelling from plain text [6]. It is specifically designed to handle large text collections, using data streaming and efficient incremental algorithms.

We made use of multiple tools from this library. We used `Dictionary` object for vocabulary filtering and embedding into the BOW representation and we used its TF-IDF implementations. We also used its API to train word vectors.

Most important for us was gensims implementation of the probabilistic algorithm for constructing approximate matrix decomposition [20]. This provides fast and easy to use incremental SVD decomposition.

3.4.1.3 Numpy

NumPy is a library for Python, adding support for large, multi-dimensional arrays and matrices [43]. We used this library for result evaluation and to implement our own logistic regression.

3.4.1.4 TensorFlow

TensorFlow is an open source software library for high performance numerical computation developed by Google Brain team [1]. It provides simple interface to matrix computations on GPUs and CPUs.

One of the most important capabilities of this library is automatic differentiation. We can create symbolic expression representing the desired computation $\hat{y} = g(X\Theta)$ and this library can automatically compute the gradient

$$\frac{\partial \hat{y}}{\Theta}$$

Because of this, TensorFlow is very popular among machine learning researchers.

We wanted to implement our whole approach in TensorFlow. It even contains implementation of SVD, however that implementation does not have all the properties that we require.

3.4.2 Other tools

During the development of required code we worked in Jupyter Notebook. It is an interactive environment the offers very easy prototyping and data visualization [49]. For data visualization we used the Matplotlib library [24]. Finally, for data manipulation and presentation we used the Pandas [40].

Last important mention belongs to Aysen Tatarinov, his blog and his repository [2], where he implemented a number of supervised weight schemes [37].

3.4.2.1 Cloud

Because we want to test multiple parameters on multiple datasets, the number of experiments we need to perform explodes. Running all those experiments on one PC would be unfeasible.

We made use of the Google Cloud service [3]. We choose Google cloud mainly because it is easy to use and it provides a free trial with 300$ of credit that can be spend on computation. This credit equals to approximately 6 moths of compute time on PC with 2 CPUs and with 7.5GB of memory (`n1-standard2`).

Thanks to a good experiment design and a few bash scripts, we were able to run experiments on 10 machines at parallel.

In the end, our experiments costed approximately 40$ worth of computation time.

[2]`https://github.com/aysent/supervised-term-weighting`

[3]`https://cloud.google.com/`

Chapter 4

Experiments

In this chapter, we present our experimental results and compare them to baseline solutions. The most important baseline that we would like to improve upon is the standard LSA. Further, we present the bag of words baseline and a baseline based on neural embeddings.

To make our evaluation more robust, we evaluate all baselines and approaches on multiple classification datasets and multiple classification tasks.

4.1 Classification datasets

To evaluate the performance of different approaches, we make use of multiple popular datasets. In this section, we describe these datasets in more detail.

4.1.1 Overview

We use datasets from SentEval[1] repository. SentEval is a library for evaluating the quality of unsupervised sentence embeddings [15]. We do not directly use any code from this library, because we were not able to modify its API in a maintainable way to fit our needs. In the end, we just use it as a guideline for dataset acquisition. Specifically, we only use the `get_transfer_data.bash` script.

This repository contains multiple different datasets, but we only use those related to binary classification tasks. Namely, we use:

- Movie review sentiment analysis dataset – MR.
- Product review dataset – CR.
- Subjectivity/objectivity dataset – SUBJ.
- Question-type dataset – TREC.
- Opinion polarity dataset – MPQA.

[1]`https://github.com/facebookresearch/SentEval`

	CR	MPQA	MR	SUBJ	TREC
#examples	3775	10606	10662	10000	5952
#unique words	5674	6238	20325	22636	8968
#words	75932	32779	230162	246015	58468
#words with 1 appearance	2714	3117	10160	11152	5338
avg sentence length	20.11	3.09	21.59	24.60	9.82
max sentence length	106	44	62	122	37
median sentence length	18	2	21	23	9
bias	0.64	0.69	0.50	0.50	NaN

Table 4.1: Dataset characteristics

In table 4.1, we present a few basic statistics for each dataset. Column *bias* denotes percentage of the majority class in the dataset. It also presents a baseline accuracy which a naive algorithm could achieve by making constant predictions. Bias for TREC dataset is not applicable because this dataset is not binary and we address this in the section 4.1.1.4.

We introduce each of these datasets in more detail and we provide some example sentences from these datasets. Note that these examples are displayed in a very genuine way (they do not contain mistakes!). We see these examples as they really are and as they are presented to our algorithms. The actual preprocessing will be discussed later in this section.

4.1.1.1 Movie review – MR

The rise of internet forums and critique websites gave birth to the problem of sentiment classification.

This dataset contains movie reviews from the site Rotten Tomatoes collected by Bo Pang and Lillian Lee [45]. Originally, each comment is accompanied by categorical rating with values “fresh” (good movie) to “rotten” (bad movie). These labels were transformed into this freely available dataset[2].

Example of positive review in this dataset is: *the rock is destined to be the 21st century 's new conan and that he 's going to make a splash even greater than arnold*

[2] `http://www.cs.cornell.edu/people/pabo/movie-review-data/`

schwarzenegger , jean-claud van damme or steven segal .

An example of a negative review is: *simplistic , silly and tedious .*

4.1.1.2 Product review – CR

This dataset was introduced by Hu and Liu [22] as a part of a customer review summarization task. They created a summarization pipeline with multiple steps and addressed a lot of different problems.

We are only interested in the sentiment prediction part, where they present a novel product review dataset.

Example of a positive review: *im a more happier person after discovering the i/p button ! .*

Example of a negative review: *weaknesses are minor : the feel and layout of the remote control are only so-so ; . it does n 't show the complete file names of mp3s with really long names ; . you must cycle through every zoom setting (2x , 3x , 4x , 1/2x , etc .) before getting back to normal size [sorry if i 'm just ignorant of a way to get back to 1x quickly] .*

4.1.1.3 Opinion polarity – MPQA

Another sentiment dataset was collected by Wiebe [59]. Its focus is to capture not only the overall tone of the document, but also strength of the tone. However, we will use only the binary positive-negative classification. Wiebe designed a fine grained annotation scheme and employed it on the sentence corpus of articles from the world press.

Example of a positive sentence: *are also being encouraged.*

Example of a negative sentence: *failing to support.*

4.1.1.4 Question-type – TREC

An important step in question answering and other dialog systems is to classify the question to the anticipated type of the answer. For example, the question of *Who is a good boy?* should be classified into the type of animal (entity), because the question probably refers to some dog. This information would narrow down the search space to identify the correct answer string [23].

This dataset contains questions and their labels in such classification. There are 6 labels:

- abbreviation (ABBR): *what is the full form of .com?*
- description (DESC): *how did serfdom develop in and then leave russia?*

- entity (ENTY): *what films featured the character popeye doyle?*
- human (HUM): *what contemptible scoundrel stole the cork from my lunch?*
- location (LOC): *what sprawling u.s. state boasts the most airports?*
- numeric (NUM): *when was ozzy osbourne born?*

As you can see, this dataset is not binary. For our purposes, we create 6 different binary datasets out of this one. To do so we employ *one-vs-all* strategy (one-vs-the-rest) used mainly to train prediction models. This strategy consists on fitting one classifier per class. For each classifier, the class is fitted against all the other classes. Hence, new dataset TREC-ABBR will contain the same samples, but labels will be binary – 1 if the sentence was originally in class ABBR and 0 otherwise.

	ABBR	DESC	ENTY	HUM	LOC	NUM
bias	0.98	0.78	0.77	0.78	0.85	0.83

Table 4.2: TREC tasks characteristics

4.1.1.5 Subjectivity/objectivity, SUBJ

There is number of sub-tasks that can help in context of sentiment prediction. One such task is to decide whether the text was subjective or objective.

Pang and Lee were able to mine the Web and create a large, automatically labeled sentence corpus. To gather subjective sentences, they collected movie review snippets from website `www.rottentomatoes.com`. To obtain objective data, they took sentences from plot summaries available from the Internet Movie Database `www.imdb.com`[3] [44].

Example of a objective review: *the movie begins in the past where a young boy named sam attempts to save celebi from a hunter .*

Example of a subjective review: *smart and alert , thirteen conversations about one thing is a small gem .*

4.1.1.6 Preprocessing

Examples provided for each dataset could be slightly strange at the first glance. We deliberately do only minimal processing in terms of word filtering and normalization. The datasets are already tokenized, we only set text to lower-case. We do not filter

[3]We personally think, that this technique is rather questionable at best, because the labels are probably very noisy.

any stop words (`the`) or non token characters (like question marks `?`). Not filtering words that are generally filtered showed to be an important decision that allowed us to get a new insight about data.

LSA is commonly accompanied by heavy filtering and it is known that stop words removal helps the performance. Our rational behind minimal filtering is that we would like the weighting scheme to understand which words are important. Hence, we hope that if the stop word is really not important, its weight will be reduced.

4.1.2 Evaluation metrics

Because we deal with binary classification tasks, our metric of choice is the accuracy.

Accuracy of p predictions in a vector $\hat{y}$, given the true labels in a vector y is computed as:

$$acc_y(\hat{y}) = \frac{1}{p} \sum_{i=1}^{p} y_i = \hat{y}_i$$

Note that this metric can have problems with unbalanced datasets. We just need to remember that what is a good accuracy depends on the dataset. For example accuracy 0.69 on MPQA dataset is really bad, because it can be achieved by constant classifier. To address this issue, we usually subtract this baseline from our results and look only on the relative improvement.

We will compute accuracy on three subsets of the dataset. acc_{train} on the train set, acc_{valid} on the validation set and acc_{test} on the test set. Note that acc_{train} is not very interesting on its own, but if we compare it with acc_{valid} and acc_{test} we can get an insight about whether our model is overfitting or underfitting.

4.2 Baselines

There is a number of baselines that we consider. They are commonly used approaches, such as pretrained neural embeddings, or very naive baselines, such as a constant classifier.

To address the problems of accuracy as a metric, we report relative improvements over the bias of each dataset. This can be viewed as an improvement over the accuracy of the constant classifier. For absolute accuracies refer to the appendix B.

The most important baseline we want to improve upon is in section 4.2.3, and it is the standard LSA.

4.2.1 Constant baseline

As discussed in the section 4.1.1, some of our datasets are biased. They may contain substantially more examples belonging to one label than belonging to the other. We can exploit this property and "train" a very simple constant classifier. This classifier counts the number of samples in each class in the training set and then always outputs the majority class.

We do not include the result table for this classifier, it is almost identical to the bias of given dataset. There are minor differences (on 3rd or 4th decimal place) because of the random data split.

4.2.2 Bag of word baselines

First, we evaluate baselines that work with local word representation (section 1.4.1). These baselines have no knowledge about semantic properties of words and usually only capture some basic word statistic. They are usually the first pick for any task, because they are simple and easy to use.

4.2.2.1 Logistic regression

First commonly used baseline is a BOW representation with logistic regression as a classifier. We test multiple different supervised and unsupervised word weighting schemes.

It could be argued that using term weights is not necessary when using logistic regression as a classifier. The argument is that the term weighs could in theory be absorbed into the weights of the classifier. This argument is actually true, if we only consider the TF part of the weighting scheme.

However, this argument only holds for the optimal solution. The problem is that even though an equivalent solution exists, it may not be found by the learning algorithm. In practice, it is useful to add the weighting scheme, as it introduces some form of knowledge (a bias), because then we can tell the algorithm which words are (probably) more important. Because of that, we may employ more strict regularization techniques without significant decrease in the performance.

scheme		CR	MPQA	MR	SUBJ
None	test	0.150	**0.150**	0.261	0.407
	train	0.333	0.224	0.477	0.495
tfchi2	test	0.114	0.119	0.184	0.336
	train	0.151	0.167	0.206	0.345
tfgr	test	0.109	0.125	0.168	0.340
	train	0.151	0.168	0.209	0.345
tfidf	test	0.134	**0.150**	0.247	0.404
	train	0.362	0.291	0.500	0.500
tfig	test	0.131	0.127	0.175	0.339
	train	0.152	0.167	0.212	0.347
tfor	test	**0.154**	**0.150**	**0.269**	**0.408**
	train	0.255	0.198	0.403	0.436
tfrf	test	0.125	0.139	0.228	0.384
	train	0.200	0.181	0.323	0.419

Table 4.3: Accuracy improvements for BOW baseline

scheme		ABBR	DESC	ENTY	HUM	LOC	NUM
None	test	**0.011**	**0.144**	0.100	0.137	0.114	0.128
	train	0.011	0.199	0.209	0.203	0.141	0.159
tfchi2	test	0.009	0.097	0.014	0.115	0.099	0.093
	train	0.009	0.091	0.013	0.112	0.103	0.088
tfgr	test	0.007	0.085	0.009	0.105	0.096	0.094
	train	0.007	0.096	0.013	0.112	0.100	0.095
tfidf	test	0.007	**0.144**	**0.110**	**0.146**	**0.120**	**0.138**
	train	0.016	0.218	0.226	0.216	0.154	0.170
tfig	test	0.007	0.091	0.009	0.104	0.103	0.092
	train	0.007	0.096	0.014	0.112	0.101	0.093
tfor	test	0.006	0.082	0.077	**0.146**	0.106	0.118
	train	0.007	0.172	0.166	0.185	0.123	0.141
tfrf	test	0.005	0.074	0.062	0.125	0.084	0.113
	train	0.007	0.144	0.132	0.159	0.105	0.126

Table 4.4: Accuracy improvements for BOW baseline on TREC datasets

According to tables 4.3 and 4.4, the best weighting schemes are `None`, `tfidf`, and `tfor`. We expect the other supervised schemes to perform much better. It is interesting, that TREC dataset is dominated by the unsupervised schemes.

4.2.3 LSA baselines

The most important baseline that we need to consider is the standard LSA. In this baseline we construct the term matrix M, reweight it with weighting scheme, train the LSA embedding and train the classifier. We try multiple weighting schemes and multiple classifiers.

See tables B.7 and B.8 (and others) in appendix B for results for dimensions 300 and 400.

		CR	MPQA	MR	SUBJ
scheme					
None	test	0.116	0.056	0.162	0.371
	train	0.160	0.069	0.196	0.387
tfchi2	test	0.111	0.082	0.166	0.333
	train	0.147	0.099	0.187	0.343
tfgr	test	0.117	0.084	0.171	0.331
	train	0.149	0.098	0.190	0.345
tfidf	test	0.115	0.063	0.183	0.390
	train	0.164	0.073	0.210	0.400
tfig	test	0.120	0.085	0.158	0.337
	train	0.148	0.098	0.192	0.345
tfor	test	0.142	0.088	0.229	0.378
	train	0.207	0.099	0.273	0.399
tfrf	test	0.115	0.094	0.176	0.374
	train	0.165	0.099	0.216	0.389

Table 4.5: Accuracy improvements for LSA baseline with 200 dimensions

Slightly unfortunate observation is that LSA performs worse than the BOW baseline. The problem is that in fact, we do not use the full potential of LSA in this baseline. The power of LSA comes from the fact that it can be trained on documents that we do not have labels for.

scheme		ABBR	DESC	ENTY	HUM	LOC	NUM
None	test	0.008	0.119	0.067	0.129	0.100	0.119
	train	0.008	0.127	0.098	0.147	0.110	0.124
tfchi2	test	0.008	0.085	0.010	0.105	0.102	0.085
	train	0.010	0.094	0.013	0.113	0.103	0.088
tfgr	test	0.007	0.096	0.013	0.111	0.096	0.102
	train	0.007	0.098	0.013	0.112	0.101	0.095
tfidf	test	0.008	0.108	0.078	0.131	0.103	0.117
	train	0.011	0.125	0.103	0.153	0.115	0.135
tfig	test	0.005	0.091	0.012	0.108	0.093	0.093
	train	0.007	0.100	0.014	0.109	0.101	0.091
tfor	test	0.008	0.083	0.076	0.133	0.102	0.126
	train	0.006	0.159	0.152	0.179	0.117	0.138
tfrf	test	0.006	0.074	0.057	0.120	0.095	0.113
	train	0.007	0.117	0.105	0.148	0.102	0.122

Table 4.6: Accuracy improvements for LSA baseline with 200 dimensions on TREC datasets

4.2.4 Neural embedding baselines

Thanks to results of Levy et al.[34] discussed in the section 1.4.5 we know that LSA is, to some extent, equivalent to neural embeddings. Because of that, we consider neural embeddings as one of our baselines. There are two possible setups: we can train our own word vectors on each dataset, or we can use pretrained embeddings that were trained on huge volumes of data.

We expect the trained embeddings to perform worse than pretrained and worst than LSA.

4.2.4.1 Trained neural embeddings

In this baseline, we train word2vec neural embeddings on the dataset we want to classify on. We use implementation `gensim.models.Word2Vec` from gensim. We train embeddings with dimensions 200, 300 and 400. Than we compute embedding for each word and sum them into a sentence embedding. Finally, we train the classifier (SVM or logistic regression) and present the accuracy. We refer to SVM as SVC (support vector classifier). Note that we train embeddings for each dataset separately.

			CR	MPQA	MR	SUBJ
logistic	200	test	0.03	0.0	0.10	0.31
		train	0.03	0.0	0.11	0.31
	300	test	0.01	0.0	0.09	0.31
		train	0.02	0.0	0.11	0.31
	400	test	0.01	0.0	0.09	0.32
		train	0.02	0.0	0.11	0.31
svc	200	test	-0.00	0.0	0.05	0.17
		train	0.00	0.0	0.08	0.19
	300	test	-0.00	0.0	0.05	0.16
		train	0.00	0.0	0.08	0.18
	400	test	-0.00	0.0	0.05	0.14
		train	0.00	0.0	0.08	0.16

Table 4.7: Accuracy improvements for trained word vectors

			ABBR	DESC	ENTY	HUM	LOC	NUM
logistic	200	test	-0.0	0.00	-0.0	0.03	0.0	0.06
		train	-0.0	0.01	-0.0	0.02	0.0	0.05
	300	test	-0.0	0.01	-0.0	0.01	-0.0	0.05
		train	-0.0	0.00	-0.0	0.01	-0.0	0.04
	400	test	-0.0	0.00	-0.0	0.00	-0.0	0.04
		train	-0.0	0.00	-0.0	0.01	-0.0	0.03
svc	200	test	-0.0	0.00	-0.0	-0.00	-0.0	0.00
		train	-0.0	-0.00	0.0	0.00	-0.0	0.00
	300	test	-0.0	0.00	-0.0	-0.00	-0.0	0.00
		train	-0.0	-0.00	0.0	0.00	-0.0	0.00
	400	test	-0.0	0.00	-0.0	-0.00	-0.0	0.00
		train	-0.0	-0.00	0.0	0.00	-0.0	0.00

Table 4.8: Accuracy improvements for trained word vectors on TREC dataset

Tables 4.7 and 4.8 contain our results. We see zero improvement of SVM classifier for TREC datasets for all embedding sizes. TREC dataset seems to be overall very challenging dataset. Interesting observation is that accuracy on this dataset decreases as we increase the embedding dimensions. The highest accuracy increase (5%) on TREC dataset is achieved by logistic regression and embeddings size 200.

We see the biggest improvement on the SUBJ dataset (31%), with no notable dependency on embedding dimension size.

However, compared to the LSA baseline, these results are significantly worse. This is consistent with findings of Altszyler et al. [3].

We also conclude that we have not observed any overfitting, as train and test accuracies are very similar.

4.2.4.2 Pretrained neural embeddings

In this experiment, we use pretrained word embeddings. We use Python library `spacy` which provides easy interface for obtaining such embeddings.

We use `spacy.load('en_vectors_web_lg')` command, which returns a pretrained model that we can query with words. Embedding obtained in this way have 300 dimensions and is trained on a large web corpus (large collection of websites). Note that because these vectors are trained on much bigger datasets, it is not a very fair comparison to other methods presented here.

		CR	MPQA	MR	SUBJ
logistic	test	0.18	0.20	0.29	0.43
	train	0.19	0.20	0.29	0.42
svc	test	-0.00	0.17	0.24	0.40
	train	0.00	0.18	0.24	0.40

Table 4.9: Accuracy improvements for pretrained word vectors

		ABBR	DESC	ENTY	HUM	LOC	NUM
logistic	test	0.01	0.08	0.07	0.14	0.10	0.07
	train	0.01	0.11	0.09	0.14	0.10	0.10
svc	test	-0.00	0.02	-0.00	0.09	0.00	0.00
	train	-0.00	0.03	0.00	0.09	0.01	0.00

Table 4.10: Accuracy improvements for pretrained word vectors on TREC dataset

We see, that these embeddings perform much better than the trained ones. We even see a significant improvements on the TREC dataset (absolute accuracy over 0.90). Interesting observation is that logistic regression performs much better than SVM. Because we are more interested in comparative and qualitative results than quantitative results, we do not spend much time on fine tuning of hyperparameters. We think that such fine tuning could increase the accuracy of each approach, but it should not change which one is better.

We also see that pretrained word vectors perform better than LSA. However, keep in mind that these representations were pretrained on huge volumes of documents. Even though LSA achieves better results than pretrained embeddings on TREC-DESC and TREC-NUM datasets. We think, that it is because these datasets are sparse and may contain a lot of words for what we do not have the pretrained embeddings.

4.3 eLSA

In this section, we present accuracy achieved by our approach.

Note that because we test on multiple datasets and we test multiple parameters, we witness a small combinatorial explosion.

4.3.1 Parameters

As mentioned in the section 3.3.3.4, there is a number of potential hyperparameters. However, we consider only the w'-learning rate β and the number of dimensions. Instead of performing an exhaustive hyperparameter search, we test multiple combinations of weighting schemes described in sections 1.4.1.2 and 1.4.1.1 on multiple datasets.

weighting schemes	`tfidf`, `tfchi2`, `tfig`, `tfgr`, `tfor`, `tfrf`, `None`
embedding size	200, 300, 400
w'-learning rates β	0.1, 0.01, 0.001

Table 4.11: Table of experiment parameters

We perform experiments with all combinations of parameters from table 4.11 on all of the 10 datasets described in section 4.1.1. We chose these values for β and embedding size, because they they are reasonable defaults commonly used in literature.

Together we need to perform and evaluate around 630 experiments.

We compare our results against LSA baseline (LSA with given weights, but without training) and report relative improvements. Results for this baseline are in section 4.2.3.

4.3.2 Batch gradient descent

We evaluate the performance of batch gradient descent optimization routine. We perform experiments for different weighting schemes, different sizes of embeddings and different learning rates on all datasets.

Our results are in the table 4.12 and 4.13. We present only results for $\beta = 0.1$, as this learning rate shows to be the best. Experiments for other learning rates can be found in appendix B.

		CR	MPQA	MR	SUBJ
scheme	lsa				
None	200	**0.01**	**0.02**	**0.06**	**0.02**
	300	**0.02**	**0.02**	**0.05**	-0.0
	400	**0.03**	**0.01**	**0.04**	**0.01**
tfchi2	200	**0.01**	0.0	**0.01**	**0.01**
	300	0.0	-0.0	**0.02**	**0.01**
	400	**0.01**	0.0	**0.03**	**0.02**
tfgr	200	**0.01**	-0.0	**0.01**	**0.02**
	300	**0.01**	-0.0	**0.01**	**0.01**
	400	**0.03**	**0.01**	**0.01**	**0.02**
tfidf	200	**0.04**	**0.06**	**0.07**	**0.01**
	300	-0.0	**0.05**	**0.05**	0.0
	400	-0.01	**0.03**	**0.02**	**0.01**
tfig	200	0.0	**0.01**	**0.01**	-0.0
	300	0.0	**0.01**	**0.01**	**0.01**
	400	**0.03**	0.0	**0.02**	**0.01**
tfor	200	**0.01**	0.0	0.0	**0.01**
	300	0.0	0.0	-0.0	0.0
	400	-0.0	**0.02**	-0.03	**0.01**
tfrf	200	**0.03**	-0.0	0.0	-0.01
	300	-0.04	**0.01**	**0.01**	0.0
	400	-0.01	**0.01**	-0.01	0.0

Table 4.12: Accuracy increase over LSA

scheme	lsa	ABBR	DESC	ENTY	HUM	LOC	NUM
None	200	0.0	**0.01**	**0.01**	-0.0	**0.01**	-0.0
	300	0.0	**0.01**	**0.01**	**0.01**	**0.01**	-0.0
	400	-0.0	**0.01**	-0.01	**0.01**	**0.02**	0.0
tfchi2	200	0.0	**0.03**	**0.03**	**0.02**	-0.01	**0.02**
	300	-0.0	**0.01**	**0.01**	-0.0	**0.01**	**0.03**
	400	0.0	-0.01	**0.02**	**0.01**	**0.01**	**0.02**
tfgr	200	0.0	-0.01	**0.01**	**0.02**	0.0	**0.01**
	300	-0.0	**0.01**	**0.01**	**0.02**	**0.01**	**0.02**
	400	0.0	**0.03**	**0.01**	**0.02**	0.0	**0.01**
tfidf	200	0.0	**0.02**	0.0	**0.01**	**0.01**	**0.01**
	300	0.0	**0.01**	**0.02**	0.0	**0.01**	**0.01**
	400	-0.0	**0.02**	**0.02**	-0.0	**0.01**	**0.01**
tfig	200	0.0	**0.02**	**0.01**	**0.01**	**0.01**	**0.01**
	300	0.0	**0.01**	**0.01**	**0.01**	**0.01**	**0.02**
	400	0.0	-0.0	**0.01**	0.0	**0.01**	**0.02**
tfor	200	0.0	**0.02**	0.0	**0.02**	**0.01**	0.0
	300	0.0	**0.03**	**0.01**	0.0	**0.01**	**0.01**
	400	0.0	**0.03**	-0.0	-0.01	**0.01**	-0.0
tfrf	200	0.0	**0.04**	**0.03**	**0.02**	**0.02**	**0.02**
	300	-0.0	**0.04**	**0.02**	**0.02**	**0.02**	0.0
	400	0.0	**0.05**	**0.04**	**0.01**	**0.01**	0.0

Table 4.13: Accuracy increase over LSA on TREC datasets

In tables 4.13 and 4.12 we see consistent improvements of eLSA over LSA. Moreover, we see that no scheme is actually locally optimal, and each scheme can be improved by performing the eLSA. There is only one dataset that we do not improve on and that is TREC-ABBR. We explain it by the fact that this dataset is extremely hard and very biased.

These results mean that all weighting schemes have some shortcomings that may manifest on some specific tasks. We try to explain this effect in the section 4.4.

During the training, we observed an interesting behaviour for `tfchi2` weighting scheme on TREC datasets. When we computed the 400 dimensional LSA embedding, it produced only 398 dimensions. This means that the original matrix was in fact just 398 dimensional, which means that the weighting scheme already filtered out a lot of information (hopefully noise) from the data.

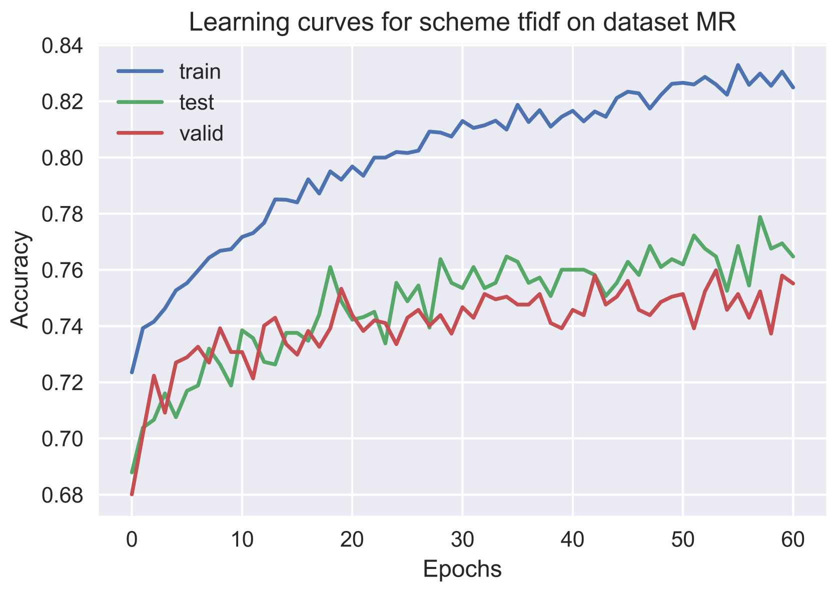

Figure 4.1: Learning curve for eLSA with tfidf weights on MR dataset

4.3.2.1 Learning curves

We examine learning curves for eLSA. In each epoch, we evaluate accuracy on training, validation and testing set.

On figure 4.1 we see a standard progress of accuracy through epochs. We observe that these curves are not entirely smooth as in batch gradient descent in neural networks. This is because we are not performing the exact gradient descent. The SVD part of eLSA is not deterministic and can be influenced by the weight w'.

Recall from section 3.3.3.2 that we monitor the validation accuracy and stop the learning process when it plateaus.

In general we observe, the number of required epochs to be between 30 and 70. The number of epochs looks to be more dependant on the dataset then on the initially weighting scheme.

4.3.2.2 Multiple gradient steps

We perform an experiment, where we do multiple gradient steps on w' for each loop. This optimization routine is illustrated by algorithm 3.

```
Data: $M$
Result: trained $W$, $S$, $C$
$w'^0 = 1$, $t = 1$;
while performance is improving on validation dataset, $t{+}{+}$ do
    recompute $S^t$ from $M \circ w'^{t-1}$;
    compute embeddings $v^t$ from $M \circ w'^{t-1}$ with $S^t$;
    fully train $C^t$ on $v^t$ and labels $y$;
    for $0 \leq i < m$ do
        update $W^t$: $w'^t = w'^{t-1} - \beta \frac{\partial E(C(S(Mw'^{t-1}))}{\partial w'^{t-1}}$
    end
end
```

Algorithm 3: stochastic training of w'

However, this introduces another hyperparameter: number of performed w' steps. We would like to avoid new hyperparameters, so we do not spend too much resources on this experiment. Second problem with this approach is that if we perform m steps, the algorithm may be m times slower. This is a serious issue for $m > 5$. We perform experiments only for weighting scheme `None` with 200 dimensional embedding and learning rate 0.1. We perform $m = 2$ and $m = 5$ gradient steps and we present results in tables 4.14, 4.15,4.16 and 4.17.

	CR	MPQA	MR	SUBJ
test	-0.01	0.00	**0.01**	**0.01**
train	0.02	0.00	0.01	-0.00
valid	-0.02	0.01	-0.01	0.00

Table 4.14: Accuracy increase for 2 steps compared to 1 step

	ABBR	DESC	ENTY	HUM	LOC	NUM
test	0.0	-0.01	-0.01	0.00	-0.01	**0.01**
train	0.0	0.00	0.01	0.00	0.00	0.00
valid	-0.0	0.02	-0.00	0.01	0.00	-0.01

Table 4.15: Accuracy increase for 2 steps compared to 1 step on TREC dataset

	CR	MPQA	MR	SUBJ
test	-0.01	**0.01**	**0.02**	**0.01**
train	0.03	0.01	0.01	0.01
valid	-0.02	0.01	0.00	-0.01

Table 4.16: Accuracy increase for 5 steps compared to 1 step

	ABBR	DESC	ENTY	HUM	LOC	NUM
test	0.0	-0.01	-0.00	-0.00	0.00	**0.01**
train	0.0	0.01	0.01	0.00	0.01	0.01
valid	0.0	0.00	0.02	-0.01	-0.00	0.01

Table 4.17: Accuracy increase for 5 steps compared to 1 step on TREC dataset

We conclude that making multiple gradient steps on w' does not significantly nor consistently improve the accuracy and only makes the training process slower. The intuition that multiple gradient steps on w' may decrease the convergence time also shows to be wrong.

	CR	MPQA	MR	SUBJ
\$m\$				
1	33	52	57	37
2	32	43	45	46
3	32	56	48	36
4	33	68	63	39
5	35	53	48	41

Table 4.18: Number of needed training steps for different m

m	ABBR	DESC	ENTY	HUM	LOC	NUM
1	32	38	38	72	38	48
2	32	39	40	45	38	36
3	32	48	35	38	32	38
4	32	34	45	40	33	47
5	32	37	36	32	37	52

Table 4.19: Number of needed training steps for different m on TREC dataset

The only relevant decrease in the number of required learning steps in tables 4.18 and 4.19 can be seen for `TREC-HUM`. However this direction does not look very promising and we do not explore it further.

4.4 Weights analysis

Most machine learning algorithm are primarily used as black box approaches [51]. It is hard, to justify their predictions or to extract some insight from them.

We designed eLSA with interpretability in mind. Because eLSA just re-weights words, we argue that it is more easily interpretable and we can extract important insight about our data from it. In this section, we present such insight.

4.4.1 Effect of reweighting

The core of our approach it to reweight the co-occurrence matrix M with weights w'. These weights effectively change importance of some words with regards to the classification task. We can directly examine them and see, what words were boosted and what words were inhibited.

Note that high word importance still does not tell us anything about the label. We do not know (at least not from w' alone), if the word is important for positive (1, positive sentiment) or negative label (0, negative sentiment).

Values in w' tell us which words are underweighted by the original weighting schemes and which are overweighted. We examine mostly the CR dataset and the TREC-DESC dataset, as eLSA shows some interesting insights particularly into these two datasets.

4.4.1.1 Most reweighted words

First, we examine the words that our approach reweighted the most (most extreme w'). We present the 10 most originally underweighted and the 10 most originally overweighted words for combinations of CR and TREC-DESC datasets and weighting schemes `None` (no scheme was used), `tfidf` (basic unsupervised scheme) and `tfig` (supervised scheme).

words	w'
slow	3.06
happy	2.65
you	2.60
works	2.51
good	2.50
bad	2.46
and	2.45
expect	2.38
pictures	2.38
highly	2.34

(a) Words with highest w'

words	w'
that	0.25
am	0.23
down	0.22
two	0.21
then	0.21
3	0.20
give	0.15
is	0.14
n	0.07
diaper	0.02

(b) Words with lowest w'

Table 4.20: Most reweighted words on CR dataset for scheme None

In table 4.20 we see that if we use weighting scheme `None` (term vector), we underweight some words that are obviously important such as `good` and `bad`. Our approach can correctly identify these words and boost their weights. In the same time, our approach can inhibit unrelated words like: `is`, `am`, or `that`. We see that using this weighting scheme (not using weighting scheme at all) is definitely not locally optimal.

In table 4.21 we display results for `tfidf`. Recall from section 1.4.1.1 that `tfidf` decreases importance of words that are common in the dataset. However, we see (and already know) that such words may be very important for the classification task. This is exactly what eLSA can find out. Our approach boosted words like `good` and `worst` that are very common and very discriminative. Moreover, our approach boosted the word `not` that is usually completely removed from the datasets, because it is considered to be a stop word. Last, but not least, eLSA almost completely inhibited the preposition `a` that almost certainly does not hold any importance for any classification task.

words	w'
not	3.25
price	2.92
good	2.57
love	2.40
if	2.34
would	2.21
worst	2.08
works	2.08
to	2.07
i	2.01

(a) Words with highest w'

words	w'
light	0.18
problems	0.17
camera	0.11
few	0.10
their	0.08
quality	0.06
battery	0.01
a	0.00
n	0.00
be	0.00

(b) Words with lowest w'

Table 4.21: Most reweighted words on CR dataset for scheme tfidf

words	w'
features	3.33
symantec	3.05
perfect	2.96
the	2.94
flaw	2.87
bit	2.83
slow	2.71
awesome	2.69
process	2.66
useless	2.64

(a) Words with highest w'

words	w'
that	0.92
is	0.88
great	0.83
't	0.83
not	0.72
very	0.62
,	0.59
and	0.52
camera	0.43
this	0.25

(b) Words with lowest w'

Table 4.22: Most reweighted words on CR dataset for scheme tfig

In table 4.22, we see results for a supervised weighting scheme `tfig`. When we look at the table b, we see that only a few fords were really overweighted by the scheme. For some reason, words: `this`, `,`, `and` have high weights and our approach identified them as not important. We see that other words are not penalized very much (0.8 or 0.9). On the other hand, it looks like this scheme undervalues some rather interesting words like `awesome` or `slow`. Interesting is a word `the`, what is always considered to be a stop word and is usually removed. However, according to our approach, it is underweighted. We think that it is because it can signify some form of superlative and hence indicate sentiment polarity.

Now we look at the TREC-DESC dataset. Recall from section 4.1.1 that in this task we have some question, and we need to predict, whether the answer to this question is a some form of a description (in contrary to some numerical answer for example).

words	w'
what	13.20
is	9.89
how	8.39
are	5.08
mean	4.86
long	4.60
why	3.96
big	3.93
reason	2.95
origin	2.89

(a) Words with highest w'

words	w'
form	0.57
name	0.55
where	0.52
language	0.51
does	0.48
of	0.46
when	0.45
and	0.20
the	0.15
,	0.07

(b) Words with lowest w'

Table 4.23: Most reweighted words on DESC dataset for scheme None

If we use just the term vectors (scheme `None`), there are again typical words that we underweight. The answer obviously depends on the interrogative pronoun such as: `who` or `what`. Also, words like `origin`, `mean`, and `reason` probably associate with some desired explanation. On the other hand, words like `does`, `of`, and `and` were identified like not important. We also see interesting decrease for words like `when` and `where`. We hypothesize that even though these words are interrogative pronouns, they are usually used to form dependent clauses that do not tell anything about the type of the desired answer. Our approach even correctly inhibited the token `,`.

Because we know that interrogative pronouns are important and abundant, we know that `tfidf` will underweight them. eLSA correctly compensated weight of these words.

words	w'
is	6.25
how	5.87
what	3.73
in	3.60
mean	3.51
of	3.10
come	3.09
long	2.96
for	2.94
the	2.39

(a) Words with highest w'

words	w'
from	0.42
its	0.41
nickname	0.38
address	0.34
abbreviation	0.32
fast	0.32
term	0.25
word	0.24
between	0.04
?	0.00

(b) Words with lowest w'

Table 4.24: Most reweighted words on DESC dataset for scheme tfidf

words	w'
is	7.69
are	4.52
what	3.52
mean	3.44
origin	3.42
difference	3.20
much	2.91
long	2.79
where	2.72
definition	2.71

(a) Words with highest w'

words	w'
out	1.00
name	0.98
you	0.97
does	0.93
in	0.90
who	0.83
do	0.71
?	0.59
was	0.46
the	0.00

(b) Words with lowest w'

Table 4.25: Most reweighted words on DESC dataset for scheme tfig

In table 4.25 we see that even supervised weighting scheme `tfig` underweights some interrogative pronouns like `what`. We see that it almost never overweights any word as only 5 words have w' lover than 0.9 and only 9 words have w' less than 1.

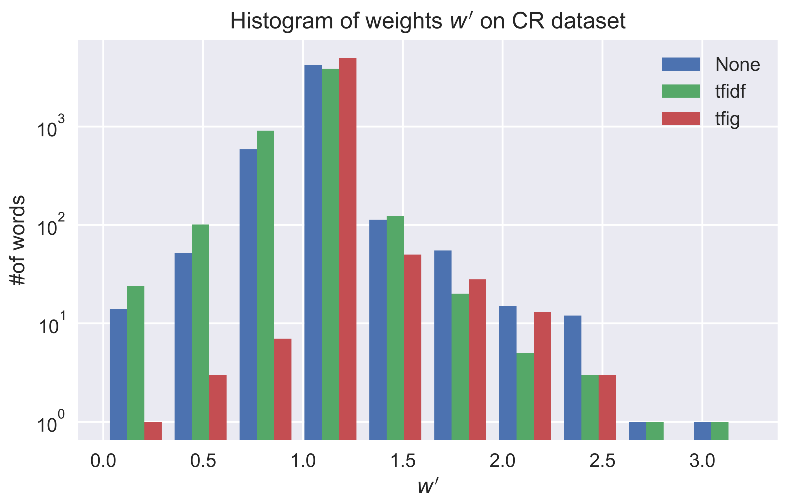

Figure 4.2: Histogram of weights w' on s CR dataset

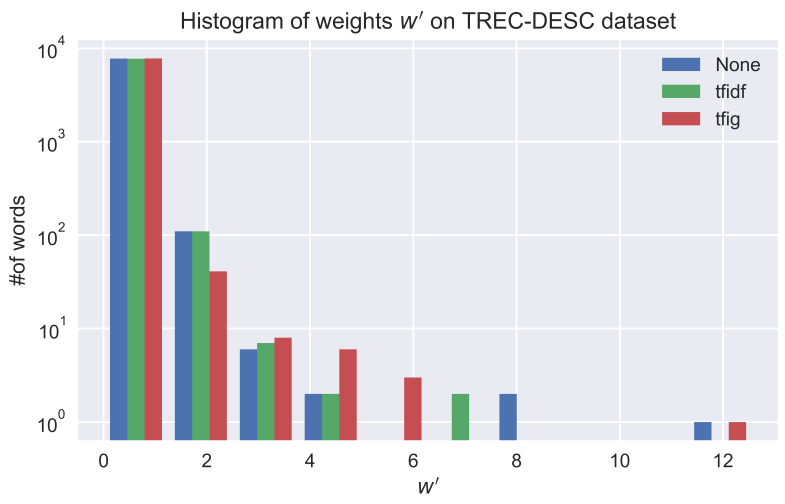

Figure 4.3: Histogram of weights w' on s TREC-DESC dataset

4.4.1.2 Weight histograms

We can look on the overall distribution of weights w' for given dataset and weighting scheme. We present histograms of values in w'. Because these values seems to be exponentially distributed, with set the y axis to logarithmic scale. This gives us an idea, whether the scheme generally underweights or overweights words. If the mass of the histogram is around 1, we know that parameters w' do not change the matrix M very much. If the mass is on the right of the 1, it means the scheme is underweighting and w' needs to compensate that. On the other hand, if the histogram is skewed to the left, it means the scheme is overweighting.

On graph 4.2 we see that `tfidf` and `None` schemes generally overweight on the CR dataset. We see `tfig` produces low values of w' less times than `None` and `tfidf`.

4.4.1.3 Possible further directions for analysis

There is a number of other interesting analysis that can be performed on parameters w' and may be subject to future work.

We think that the most interesting, and possibly the hardest task, would be to find the analytic formula for such weighted scheme that incorporates the learned parameters w'. This could lead to a design of a new supervised weighting scheme with better performance than current ones.

To learn such analytic formula, we recommend to use symbolic regression.

4.5 Discussion

We conclude that eLSA can improve performance of all tested schemes. For some combination of schemes and datasets we saw a major increase (`tfidf` on MR dataset gained 7%), and for some (`ftrf` on CR dataset lost 4%) we see a decrease in the achieved accuracy. We have to say, that eLSA can overfit the training dataset. This is naturally caused by the number of trainable parameters w' that we introduce. Size of w' is the same as size of the vocabulary $|V|$, and it ranges from $32,000$ to $240,000$ in our datasets.

Because eLSA introduces only one hyperparameter (w'-learning rate β), it is super easy to use. However, running eLSA is much more time consuming compared to standard LSA. The problem is, that we in fact perform one whole LSA for each epoch of eLSA. From our observations the number of required epochs ranges from 30 to 70 with some outliers reaching as much as 97. This can make eLSA unsuitable for some use cases, but we think that it can always be used as a diagnostic tool.

Indeed we successfully used eLSA to diagnose problems of given weighting schemes. Words that eLSA boosted seemed to be reasonable and we were able to gain insight into the datasets. We found, that the word `the` may be very important for certain tasks, even though it is almost always removed as a part of the data preprocessing. Furthermore eLSA showed us how important interrogative pronouns can be on TREC dataset.

We need to point out, that we explored only a small subspace of eLSA hyperparameters. Despite that, we achieved significant improvements in accuracy. More effort can be put into exploring this hyperparameter space in the future work.

Conclusion

In this thesis, we revisited a famous co-occurrence-based approach called LSA. First, we experimentally confirmed that using LSA yields higher accuracy than using word vectors (word2vec) trained on the same corpus.

To improve the LSA even further, we described a novel way of how to incorporate the supervised information into the LSA. We designed eLSA, method that incorporates knowledge of the classifier's error directly to the weighting scheme. eLSA introduces a new layer of reweighting that is applied before the SVD decomposition. This layer allows the unsupervised SVD process to become supervised. We have experimentally shown, that it is possible to train eLSA with batch gradient descent and that the training error nicely converges to local optima.

We evaluated eLSA for combinations of 10 classification datasets and 7 weighting schemes. In the most of these experiments, eLSA consistently achieved higher accuracy than the LSA baseline. We have confirmed that none of the used schemes is optimal on all datasets and hence their design can be further improved. eLSA was able to improve the accuracy achieved by these schemes by up to 7%.

Very important and useful property of eLSA is its interpretability. We analyzed parameters w' learned by eLSA and showed that they can be used to gain insight into the dataset. Based on that we identified multiple groups of words that are commonly underweighted by the weighting schemes. Words such as interrogative pronoun showed to be usually underweighted in the question classification task (TREC-DESC), even by supervised weighting schemes. We do not want to underweight such important words, because we do not want the SVD to filer them as a noise. To our surprise, words that are considered to be stop words (like `the` or `not`) and are usually entirely removed from the dataset, also showed to be underweighted. We hypothesize that this word is actually really important, because it may signal superlatives and hence opinion polarity.

To conclude, we fulfilled all our goals.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Akiko Aizawa. An information-theoretic perspective of tfidf measures. *Information Processing & Management*, 39(1):45–65, 2003.

[3] Edgar Altszyler, Mariano Sigman, Sidarta Ribeiro, and Diego Fernández Slezak. Comparative study of lsa vs word2vec embeddings in small corpora: a case study in dreams database. *arXiv preprint arXiv:1610.01520*, 2016.

[4] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 238–247, 2014.

[5] Michael W Berry, Susan T Dumais, and Gavin W O'Brien. Using linear algebra for intelligent information retrieval. *SIAM review*, 37(4):573–595, 1995.

[6] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python.* " O'Reilly Media, Inc.", 2009.

[7] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[8] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural*

Information Processing Systems, volume 20, pages 161–168. NIPS Foundation (http://books.nips.cc), 2008.

[9] Vladimír Boža. personal communication.

[10] Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30, 2006.

[11] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[12] Sutanu Chakraborti, Robert Lothian, Nirmalie Wiratunga, and Stuart Watt. Sprinkling: supervised latent semantic indexing. In *European Conference on Information Retrieval*, pages 510–514. Springer, 2006.

[13] Sutanu Chakraborti, Rahman Mukras, Robert Lothian, Nirmalie Wiratunga, Stuart NK Watt, and David J Harper. Supervised latent semantic indexing using adaptive sprinkling. In *IJCAI*, volume 7, pages 1582–1587, 2007.

[14] Wikimedia Commons. Illustration of gradient descent on a series of level sets. `"https://en.wikipedia.org/wiki/Gradient_descent#/media/File:Gradient_descent.svg"`, 2012.

[15] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*, 2017.

[16] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[17] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990.

[18] Zhi-Hong Deng, Kun-Hu Luo, and Hong-Liang Yu. A study of supervised term weighting scheme for sentiment analysis. *Expert Systems with Applications*, 41(7):3506–3513, 2014.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[20] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[21] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[22] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.

[23] Zhiheng Huang, Marcus Thint, and Zengchang Qin. Question classification using head words and their hypernyms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 927–936. Association for Computational Linguistics, 2008.

[24] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[25] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *arXiv preprint arXiv:1509.07838*, 2015.

[26] Yangfeng Ji and Jacob Eisenstein. Discriminative improvements to distributional sentence similarity. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 891–896, 2013.

[27] Daniel Jurafsky. Speech and language processing: An introduction to natural language processing. *Computational linguistics, and speech recognition*, 2000.

[28] April Kontostathis and William M Pottenger. A framework for understanding latent semantic indexing (lsi) performance. *Information Processing & Management*, 42(1):56–73, 2006.

[29] Man Lan, Chew Lim Tan, Jian Su, and Yue Lu. Supervised and traditional term weighting methods for automatic text categorization. *IEEE transactions on pattern analysis and machine intelligence*, 31(4):721–735, 2009.

[30] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[31] Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*, 2014.

[32] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1188–1196, 2014.

[33] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[34] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[35] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[36] Hang Li, Jun Xu, et al. Semantic matching in search. *Foundations and Trends® in Information Retrieval*, 7(5):343–469, 2014.

[37] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

[38] Vladimír Macko. Dolovanie súvisiacich patentov k vedeckým článkom. Bachelor thesis, Comenius university, Bratislava, 2016.

[39] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[40] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[41] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[42] Marwa Naili, Anja Habacha Chaibi, and Henda Hajjami Ben Ghezala. Comparative study of word embedding methods in topic segmentation. *Procedia Computer Science*, 112:340–349, 2017.

[43] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[44] Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics, 2004.

[45] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[46] Christos H Papadimitriou, Prabhakar Raghavan, Hisao Tamaki, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. *Journal of Computer and System Sciences*, 61(2):217–235, 2000.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[48] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[49] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.

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

[51] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

[52] Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

[53] Herbert Rubenstein and John B. Goodenough. Contextual correlates of synonymy. *Commun. ACM*, 8(10):627–633, October 1965.

[54] Herbert Rubenstein and John B Goodenough. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633, 1965.

[55] DE Rumelhart. Learning internal representations by error propagation. *Nature*, 323:533–536, 1986.

[56] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[57] Jian-Tao Sun, Zheng Chen, Hua-Jun Zeng, Yu-Chang Lu, Chun-Yi Shi, and Wei-Ying Ma. Supervised latent semantic indexing for document categorization. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 535–538. IEEE, 2004.

[58] Bc. Mária Vajdová. Vektorová reprezentácia slov využívajúca morfológiu. Master's thesis, Comenius university, Bratislava, 2017.

[59] Janyce Wiebe, Theresa Wilson, and Claire Cardie. Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 39(2-3):165–210, 2005.

[60] Peter Wiemer-Hastings, K Wiemer-Hastings, and A Graesser. Latent semantic analysis. In *Proceedings of the 16th international joint conference on Artificial intelligence*, pages 1–14. Citeseer, 2004.

[61] Haibing Wu, Xiaodong Gu, and Yiwei Gu. Balancing between over-weighting and under-weighting in supervised term weighting. *Information Processing & Management*, 53(2):547–557, 2017.

[62] Mo Yu and Mark Dredze. Learning composition models for phrase embeddings. *Transactions of the Association for Computational Linguistics*, 3:227–242, 2015.

Appendix A

Source code

Source code to this diploma thesis with installation guide is available on GitHub: `https://github.com/vlejd/eLSA`

Documentation also contains guide for extending the evaluation for further datasets, test different weighting schemes or different classifiers.

Appendix B

Detailed results

scheme		CR	MPQA	MR	SUBJ
None	test	0.787	0.838	0.761	0.907
	train	0.971	0.912	0.977	0.995
tfchi2	test	0.751	0.806	0.684	0.836
	train	0.789	0.854	0.706	0.845
tfgr	test	0.747	0.813	0.668	0.840
	train	0.789	0.856	0.709	0.845
tfidf	test	0.771	0.838	0.747	0.904
	train	0.999	0.979	1.000	1.000
tfig	test	0.769	0.815	0.675	0.839
	train	0.790	0.854	0.712	0.847
tfor	test	0.792	0.838	0.769	0.908
	train	0.893	0.886	0.903	0.936
tfrf	test	0.762	0.827	0.728	0.884
	train	0.837	0.869	0.823	0.919

Table B.1: Accuracy for BOW baseline

scheme		ABBR	DESC	ENTY	HUM	LOC	NUM
None	test	0.995	0.925	0.874	0.920	0.960	0.959
	train	0.995	0.981	0.983	0.986	0.987	0.990
tfchi2	test	0.993	0.878	0.789	0.899	0.945	0.923
	train	0.993	0.872	0.787	0.895	0.949	0.919
tfgr	test	0.991	0.866	0.784	0.889	0.942	0.924
	train	0.991	0.878	0.788	0.896	0.946	0.926
tfidf	test	0.991	0.925	0.885	0.930	0.966	0.968
	train	1.000	1.000	1.000	1.000	1.000	1.000
tfig	test	0.991	0.872	0.784	0.887	0.949	0.922
	train	0.991	0.878	0.788	0.896	0.947	0.924
tfor	test	0.990	0.864	0.852	0.930	0.952	0.948
	train	0.991	0.954	0.940	0.969	0.969	0.971
tfrf	test	0.989	0.855	0.837	0.908	0.930	0.943
	train	0.991	0.926	0.906	0.942	0.951	0.957

Table B.2: Accuracy for BOW baseline on TREC datasets

		CR	MPQA	MR	SUBJ
scheme					
None	test	0.753	0.744	0.662	0.871
	train	0.798	0.757	0.696	0.887
tfchi2	test	0.749	0.770	0.666	0.833
	train	0.784	0.787	0.687	0.843
tfgr	test	0.754	0.772	0.671	0.831
	train	0.787	0.785	0.690	0.845
tfidf	test	0.753	0.750	0.683	0.890
	train	0.801	0.761	0.710	0.900
tfig	test	0.758	0.772	0.658	0.837
	train	0.785	0.785	0.692	0.845
tfor	test	0.780	0.776	0.729	0.878
	train	0.844	0.787	0.773	0.899
tfrf	test	0.753	0.782	0.676	0.874
	train	0.802	0.787	0.716	0.889

Table B.3: Accuracy for LSA baseline with 200 dimensions

		CR	MPQA	MR	SUBJ
scheme					
None	test	0.760	0.763	0.692	0.893
	train	0.822	0.783	0.721	0.900
tfchi2	test	0.763	0.785	0.679	0.829
	train	0.786	0.803	0.696	0.843
tfgr	test	0.765	0.788	0.666	0.842
	train	0.787	0.802	0.698	0.845
tfidf	test	0.778	0.763	0.703	0.890
	train	0.831	0.788	0.739	0.912
tfig	test	0.751	0.780	0.668	0.839
	train	0.789	0.804	0.698	0.847
tfor	test	0.798	0.781	0.744	0.892
	train	0.854	0.805	0.792	0.906
tfrf	test	0.776	0.783	0.691	0.867
	train	0.809	0.802	0.733	0.895

Table B.4: Accuracy for LSA baseline with 300 dimensions

		CR	MPQA	MR	SUBJ
scheme					
None	test	0.752	0.775	0.712	0.887
	train	0.845	0.799	0.747	0.911
tfchi2	test	0.760	0.791	0.680	0.834
	train	0.788	0.813	0.699	0.846
tfgr	test	0.754	0.790	0.680	0.827
	train	0.790	0.815	0.702	0.848
tfidf	test	0.767	0.784	0.731	0.892
	train	0.851	0.802	0.758	0.918
tfig	test	0.742	0.789	0.676	0.838
	train	0.790	0.813	0.702	0.846
tfor	test	0.793	0.792	0.756	0.879
	train	0.863	0.820	0.803	0.912
tfrf	test	0.765	0.795	0.706	0.871
	train	0.815	0.814	0.747	0.901

Table B.5: Accuracy for LSA baseline with 400 dimensions

scheme		CR	MPQA	MR	SUBJ
None	test	0.116	0.056	0.162	0.371
	train	0.160	0.069	0.196	0.387
tfchi2	test	0.111	0.082	0.166	0.333
	train	0.147	0.099	0.187	0.343
tfgr	test	0.117	0.084	0.171	0.331
	train	0.149	0.098	0.190	0.345
tfidf	test	0.115	0.063	0.183	0.390
	train	0.164	0.073	0.210	0.400
tfig	test	0.120	0.085	0.158	0.337
	train	0.148	0.098	0.192	0.345
tfor	test	0.142	0.088	0.229	0.378
	train	0.207	0.099	0.273	0.399
tfrf	test	0.115	0.094	0.176	0.374
	train	0.165	0.099	0.216	0.389

Table B.6: Accuracy improvements for LSA baseline with 200 dimensions

scheme		CR	MPQA	MR	SUBJ
None	test	0.122	0.075	0.192	0.393
	train	0.185	0.095	0.221	0.400
tfchi2	test	0.125	0.097	0.179	0.329
	train	0.149	0.115	0.196	0.343
tfgr	test	0.127	0.100	0.166	0.342
	train	0.149	0.114	0.198	0.345
tfidf	test	0.141	0.075	0.203	0.390
	train	0.193	0.100	0.239	0.412
tfig	test	0.113	0.092	0.168	0.339
	train	0.151	0.116	0.198	0.347
tfor	test	0.160	0.094	0.244	0.392
	train	0.216	0.117	0.292	0.406
tfrf	test	0.138	0.095	0.191	0.367
	train	0.172	0.114	0.233	0.395

Table B.7: Accuracy improvements for LSA baseline with 300 dimensions

scheme		CR	MPQA	MR	SUBJ
None	test	0.114	0.088	0.212	0.387
	train	0.207	0.111	0.247	0.411
tfchi2	test	0.123	0.103	0.180	0.334
	train	0.150	0.125	0.199	0.346
tfgr	test	0.116	0.102	0.180	0.327
	train	0.152	0.127	0.202	0.348
tfidf	test	0.129	0.096	0.231	0.392
	train	0.213	0.114	0.258	0.418
tfig	test	0.105	0.102	0.176	0.338
	train	0.152	0.125	0.202	0.346
tfor	test	0.156	0.104	0.256	0.379
	train	0.226	0.132	0.303	0.412
tfrf	test	0.127	0.107	0.206	0.371
	train	0.178	0.127	0.247	0.401

Table B.8: Accuracy improvements for LSA baseline with 400 dimensions

scheme		ABBR	DESC	ENTY	HUM	LOC	NUM
None	test	0.992	0.900	0.841	0.912	0.946	0.949
	train	0.992	0.909	0.872	0.930	0.956	0.954
tfchi2	test	0.992	0.866	0.785	0.889	0.948	0.915
	train	0.994	0.875	0.787	0.897	0.949	0.919
tfgr	test	0.991	0.877	0.787	0.895	0.942	0.933
	train	0.991	0.879	0.787	0.895	0.947	0.926
tfidf	test	0.992	0.890	0.853	0.914	0.949	0.948
	train	0.995	0.906	0.877	0.936	0.961	0.966
tfig	test	0.989	0.872	0.787	0.891	0.939	0.923
	train	0.991	0.881	0.788	0.893	0.947	0.921
tfor	test	0.992	0.864	0.850	0.916	0.948	0.956
	train	0.990	0.941	0.926	0.963	0.963	0.968
tfrf	test	0.990	0.856	0.831	0.903	0.941	0.943
	train	0.991	0.899	0.880	0.932	0.948	0.952

Table B.9: Accuracy for LSA baseline with 200 dimensions on TREC datasets

scheme		ABBR	DESC	ENTY	HUM	LOC	NUM
None	test	0.992	0.906	0.847	0.917	0.953	0.955
	train	0.993	0.922	0.885	0.944	0.959	0.963
tfchi2	test	0.994	0.884	0.787	0.895	0.944	0.921
	train	0.992	0.879	0.789	0.896	0.949	0.918
tfgr	test	0.992	0.869	0.786	0.886	0.944	0.926
	train	0.991	0.880	0.787	0.896	0.949	0.923
tfidf	test	0.993	0.903	0.859	0.925	0.951	0.954
	train	0.995	0.923	0.896	0.954	0.970	0.975
tfig	test	0.990	0.876	0.787	0.886	0.941	0.925
	train	0.991	0.879	0.787	0.895	0.948	0.930
tfor	test	0.990	0.865	0.857	0.924	0.955	0.957
	train	0.991	0.943	0.931	0.966	0.965	0.971
tfrf	test	0.990	0.865	0.832	0.910	0.942	0.949
	train	0.991	0.906	0.887	0.935	0.949	0.953

Table B.10: Accuracy for LSA baseline with 300 dimensions on TREC datasets

scheme		ABBR	DESC	ENTY	HUM	LOC	NUM
None	test	0.992	0.913	0.865	0.920	0.943	0.956
	train	0.994	0.932	0.900	0.948	0.965	0.967
tfchi2	test	0.993	0.879	0.785	0.895	0.945	0.915
	train	0.993	0.879	0.787	0.898	0.948	0.917
tfgr	test	0.987	0.875	0.786	0.891	0.945	0.925
	train	0.992	0.881	0.787	0.896	0.948	0.928
tfidf	test	0.994	0.905	0.859	0.923	0.953	0.957
	train	0.995	0.933	0.909	0.965	0.980	0.985
tfig	test	0.990	0.878	0.785	0.895	0.944	0.922
	train	0.991	0.880	0.787	0.896	0.947	0.923
tfor	test	0.993	0.867	0.847	0.928	0.949	0.959
	train	0.991	0.946	0.934	0.965	0.965	0.970
tfrf	test	0.991	0.863	0.832	0.917	0.944	0.946
	train	0.991	0.906	0.887	0.936	0.950	0.954

Table B.11: Accuracy for LSA baseline with 400 dimensions on TREC datasets

scheme		ABBR	DESC	ENTY	HUM	LOC	NUM
None	test	0.008	0.119	0.067	0.129	0.100	0.119
	train	0.008	0.127	0.098	0.147	0.110	0.124
tfchi2	test	0.008	0.085	0.010	0.105	0.102	0.085
	train	0.010	0.094	0.013	0.113	0.103	0.088
tfgr	test	0.007	0.096	0.013	0.111	0.096	0.102
	train	0.007	0.098	0.013	0.112	0.101	0.095
tfidf	test	0.008	0.108	0.078	0.131	0.103	0.117
	train	0.011	0.125	0.103	0.153	0.115	0.135
tfig	test	0.005	0.091	0.012	0.108	0.093	0.093
	train	0.007	0.100	0.014	0.109	0.101	0.091
tfor	test	0.008	0.083	0.076	0.133	0.102	0.126
	train	0.006	0.159	0.152	0.179	0.117	0.138
tfrf	test	0.006	0.074	0.057	0.120	0.095	0.113
	train	0.007	0.117	0.105	0.148	0.102	0.122

Table B.12: Accuracy improvements for LSA baseline with 200 dimensions on TREC datasets

scheme		ABBR	DESC	ENTY	HUM	LOC	NUM
None	test	0.008	0.124	0.073	0.134	0.107	0.124
	train	0.009	0.140	0.110	0.160	0.113	0.132
tfchi2	test	0.010	0.103	0.012	0.111	0.098	0.091
	train	0.008	0.098	0.015	0.112	0.103	0.087
tfgr	test	0.008	0.088	0.012	0.102	0.098	0.096
	train	0.007	0.098	0.012	0.113	0.103	0.093
tfidf	test	0.009	0.121	0.085	0.142	0.105	0.123
	train	0.011	0.142	0.121	0.170	0.124	0.144
tfig	test	0.006	0.095	0.013	0.102	0.095	0.094
	train	0.007	0.097	0.012	0.112	0.102	0.099
tfor	test	0.006	0.084	0.083	0.140	0.109	0.127
	train	0.007	0.162	0.156	0.182	0.119	0.140
tfrf	test	0.006	0.084	0.058	0.127	0.096	0.119
	train	0.007	0.124	0.113	0.151	0.103	0.122

Table B.13: Accuracy improvements for LSA baseline with 300 dimensions on TREC datasets

scheme		ABBR	DESC	ENTY	HUM	LOC	NUM
None	test	0.008	0.132	0.091	0.136	0.097	0.125
	train	0.010	0.150	0.126	0.165	0.118	0.137
tfchi2	test	0.009	0.097	0.010	0.111	0.099	0.084
	train	0.009	0.097	0.013	0.114	0.102	0.086
tfgr	test	0.003	0.094	0.012	0.107	0.099	0.094
	train	0.007	0.099	0.013	0.113	0.102	0.098
tfidf	test	0.010	0.124	0.085	0.140	0.107	0.127
	train	0.011	0.151	0.135	0.181	0.134	0.155
tfig	test	0.006	0.096	0.010	0.111	0.098	0.091
	train	0.007	0.098	0.013	0.112	0.101	0.093
tfor	test	0.009	0.086	0.073	0.144	0.103	0.129
	train	0.007	0.165	0.159	0.182	0.119	0.139
tfrf	test	0.007	0.082	0.058	0.134	0.098	0.115
	train	0.007	0.124	0.113	0.152	0.104	0.124

Table B.14: Accuracy improvements for LSA baseline with 400 dimensions on TREC datasets

scheme	lsa	CR	MPQA	MR	SUBJ
None	200	-0.01	**0.01**	**0.03**	**0.01**
	300	**0.02**	-0.0	**0.03**	-0.01
	400	**0.01**	**0.01**	**0.02**	-0.0
tfchi2	200	-0.01	0.0	0.0	**0.01**
	300	-0.02	0.0	-0.0	**0.01**
	400	-0.01	**0.01**	-0.0	0.0
tfgr	200	**0.02**	-0.0	**0.02**	**0.01**
	300	-0.02	-0.0	**0.03**	**0.01**
	400	0.0	**0.01**	-0.01	**0.01**
tfidf	200	0.0	**0.01**	**0.05**	0.0
	300	-0.01	**0.02**	**0.03**	**0.01**
	400	0.0	**0.01**	**0.02**	**0.01**
tfig	200	**0.02**	-0.01	**0.02**	**0.01**
	300	-0.0	0.0	**0.01**	-0.0
	400	**0.02**	**0.01**	**0.01**	0.0
tfor	200	0.0	-0.0	**0.01**	**0.01**
	300	-0.02	**0.02**	-0.02	0.0
	400	-0.01	0.0	-0.01	**0.02**
tfrf	200	-0.01	-0.01	0.0	-0.0
	300	-0.0	0.0	-0.0	0.0
	400	-0.0	**0.01**	0.0	0.0

Table B.15: Accuracy increase over LSA for $\beta = 0.01$

scheme	lsa	ABBR	DESC	ENTY	HUM	LOC	NUM
None	200	-0.0	**0.02**	**0.01**	0.0	**0.01**	-0.0
	300	0.0	-0.0	**0.01**	-0.01	0.0	-0.0
	400	-0.0	0.0	-0.0	-0.01	**0.02**	-0.01
tfchi2	200	0.0	**0.01**	0.0	**0.01**	**0.01**	**0.01**
	300	-0.0	-0.0	**0.01**	0.0	0.0	**0.01**
	400	0.0	-0.01	**0.01**	**0.01**	0.0	**0.01**
tfgr	200	-0.0	0.0	0.0	-0.0	**0.01**	**0.01**
	300	-0.0	**0.03**	-0.0	**0.01**	-0.0	0.0
	400	**0.01**	**0.01**	-0.0	**0.01**	**0.01**	0.0
tfidf	200	0.0	**0.01**	-0.0	**0.02**	0.0	**0.01**
	300	0.0	-0.01	**0.01**	-0.01	0.0	**0.01**
	400	-0.0	-0.0	-0.01	0.0	**0.01**	-0.0
tfig	200	0.0	**0.02**	0.0	**0.01**	0.0	0.0
	300	0.0	0.0	0.0	**0.01**	-0.0	-0.0
	400	0.0	-0.01	-0.0	-0.0	-0.0	**0.01**
tfor	200	0.0	**0.01**	-0.0	**0.01**	0.0	-0.0
	300	0.0	**0.02**	-0.0	-0.0	0.0	-0.01
	400	0.0	**0.05**	**0.01**	-0.0	**0.01**	-0.01
tfrf	200	0.0	**0.04**	**0.01**	**0.01**	0.0	**0.01**
	300	0.0	**0.05**	**0.01**	0.0	-0.0	-0.0
	400	0.0	**0.04**	**0.02**	0.0	**0.01**	-0.0

Table B.16: Accuracy increase over LSA for $\beta = 0.01$ on TREC datasets

scheme	lsa	CR	MPQA	MR	SUBJ
None	200	-0.01	**0.01**	**0.02**	0.0
	300	0.0	0.0	-0.01	-0.01
	400	**0.01**	-0.01	**0.01**	**0.01**
tfchi2	200	**0.02**	0.0	-0.02	0.0
	300	-0.01	0.0	-0.01	0.0
	400	-0.01	-0.0	-0.02	0.0
tfgr	200	-0.01	**0.01**	-0.01	0.0
	300	-0.0	0.0	**0.01**	-0.01
	400	0.0	0.0	-0.01	**0.02**
tfidf	200	**0.01**	**0.01**	0.0	0.0
	300	-0.01	**0.01**	**0.01**	-0.01
	400	**0.01**	0.0	-0.01	**0.01**
tfig	200	**0.01**	**0.01**	-0.01	-0.0
	300	-0.0	0.0	**0.01**	**0.01**
	400	**0.02**	-0.0	0.0	-0.02
tfor	200	**0.01**	-0.0	-0.0	0.0
	300	-0.02	0.0	-0.01	-0.01
	400	0.0	0.0	-0.01	**0.02**
tfrf	200	**0.02**	-0.0	-0.01	0.0
	300	0.0	0.0	0.0	0.0
	400	-0.01	-0.0	-0.0	0.0

Table B.17: Accuracy increase over LSA for $\beta = 0.001$

scheme	lsa	ABBR	DESC	ENTY	HUM	LOC	NUM
None	200	-0.0	-0.01	0.0	0.0	-0.0	-0.01
	300	0.0	-0.01	0.0	-0.0	0.0	-0.01
	400	0.0	**0.01**	-0.01	0.0	0.0	-0.0
tfchi2	200	0.0	**0.01**	-0.0	**0.01**	0.0	**0.02**
	300	-0.0	-0.01	-0.0	0.0	-0.0	0.0
	400	-0.0	0.0	0.0	0.0	**0.01**	**0.01**
tfgr	200	-0.0	-0.01	-0.0	0.0	-0.0	-0.0
	300	0.0	**0.01**	-0.0	**0.01**	0.0	-0.01
	400	0.0	-0.01	-0.0	-0.01	**0.01**	-0.01
tfidf	200	-0.0	-0.0	-0.0	0.0	0.0	0.0
	300	0.0	-0.0	**0.01**	-0.01	-0.01	0.0
	400	-0.0	-0.0	**0.01**	-0.0	-0.0	0.0
tfig	200	0.0	-0.0	-0.0	-0.0	**0.01**	**0.01**
	300	0.0	-0.0	-0.0	**0.01**	0.0	-0.0
	400	-0.0	-0.01	0.0	-0.01	-0.0	-0.0
tfor	200	-0.0	-0.0	0.0	**0.01**	0.0	-0.01
	300	0.0	-0.0	0.0	-0.01	-0.0	-0.01
	400	-0.0	0.0	**0.01**	-0.0	0.0	-0.01
tfrf	200	0.0	**0.05**	**0.01**	**0.01**	-0.01	**0.01**
	300	0.0	**0.03**	-0.0	0.0	0.0	0.0
	400	0.0	**0.03**	**0.02**	0.0	-0.01	-0.0

Table B.18: Accuracy increase over LSA for $\beta = 0.001$ on TREC datasets