Institute of Natural Language Processing

University of Stuttgart
Pfaffenwaldring 5b
D–70569 Stuttgart

Masterarbeit

How Word-Embedding methods improve information extraction and can be used for multilingual approaches

Ulrich Z.

Course of Study:	Informatik
Examiner:	Prof. Dr. Sebastian Padó
Supervisor:	Dr. Yifan He
Commenced:	August 1, 2017
Completed:	February 1, 2018
CR-Classification:	I.2.7

1 Abstract

Expanding entity sets and extracting relations are key tasks in natural language processing (NLP), which is accomplished in various approaches. Recent successful attempts are all using word-embeddings like the ones presented by Mikolov et al. While most work concentrates on how to improve these tasks in general without considering a specific domain, it is of interest how to achieve even higher precisions when focusing on a specific domain and optimizing the methods towards a single purpose.
Therefore this thesis suggests methods and adjustments to optimize the proposals for entity set expansion for the domain of drugs.
While this is the main purpose of this thesis, it will also present a novel idea, how to improve the precision in relation extraction by using word-embeddings, which could be combined with existing successful relation extraction methods. And finally another key aspect of many international companies is tagged, by presenting a solution for multilingual information extraction system (IES), which is capable of preprocessing text of multiple languages, expanding entity sets independent of the language used and extracting relations on the texts.

Contents

2 Acknowledgement

First of all, I would like to thank Prof. Dr. Sebastian Padó for being open to a cooperation between University of Stuttgart and Robert Bosch LLC, and supporting an external master thesis while also assuring the scientific focus is kept.

Secondly, I want to thank Dr. Yifan He, who did an amazing job as my supervisor: giving me advice, support, and insight in the work of a research scientist at Robert Bosch LLC. I really appreciate it and hope to stay in contact beyond the admission of this thesis.

Thirdly, I also have to thank Robert Bosch GmbH and Robert Bosch LLC for offering me the opportunity to work in their departments and gaining experience throughout my studies and as part of my master thesis.

Finally, I have to thank my family, who were always supporting me with every path I chose and every decision I made, I couldn't have done it without them, and I am very grateful for having them around me!

3 Introduction

It was Sir Francis Bacon, who, in his 1597 authorship of 'ipsa scientia potestas est'[BM57], realized that knowledge is one of the most powerful sources in the world. Knowledge on the other hand has to be retrieved from the available information to be of use. While there is huge amounts of data available online or as collections saved by companies, the information in these storages is mostly unstructured and inaccessible without a lot of effort. Nowadays companies realize, that they possess huge collections of data and need to extract the information out of the data to gain knowledge from that. For this purpose information extraction is used.

Information extraction can be used for several purposes. Examples include opinion mining, question answering systems, semantic search or market analysis. Take aspect-level opinion mining from product reviews for example. If the government wants to know which and how many crimes were related to a certain drug within a specific year, there are several ways to retrieve that information. One possibility would be to define a set of questions with predefined answers and ask these questions to the judges or prosecutors. Predefined answers are simple to complete, thus making it possible to distribute the questionnaire to many people. On the other hand, predefined answers restrict the interviewed to these answers and information might get lost.
Another option would be to ask people a set of questions and give the people the liberty to answer the questions freely. These answers can be read and transformed into knowledge manually. This process, however, is time consuming and expensive as all the text has to be analyzed and evaluated. Therefore only a small group of people can be interviewed.
Instead of reading the answers manually, it is also possible to extract information automatically from free, unstructured text by using NLP techniques. The automatic extraction of information from free text is advantageous. NLP techniques permit a large group of people can be interviewed and the given data can be analyzed fast and cheap. Also in most other information extraction use cases, the NLP approach offers the most advantages and is explored to deliver increasingly more successful results.

In this thesis the focus is centered around information extraction, which extracts useful information from unstructured data. To extract information from unstructured data, the data has to be preprocessed and each word has to be represented by a data structure.

For calculating the representation of words, word-embedding methods are applied to the given data/corpus. These methods create a multi-dimensional vector for each word, which represents the words' meaning within the corpus. By using word-embedding methods, words can be represented through high-dimensional vectors and different words can be compared by calculating the similarity of their vectors. The original system uses sparse vectors, which are created by comparing the word/entity represented by the vector to any other entity in the corpus.
This thesis wants to show that using word-embedding methods is of advantage and simultaneously demonstrates how it it can be applied.
Furthermore this thesis also wants to show how information extraction can be used in a multilingual approach. This is essential for large companies, who collect data in more than one country.

Therefore the proposal of this thesis is: How do word-embedding methods improve information extraction on unstructured text and how can it be used in a multilingual approach? This main question will be split into two subquestions:

1. Can Mikolov-style word-embeddings improve the entity set expansion method of Min and Grishman?
2. How can Mikolov-style word-embeddings be used to improve relation extraction based on syntactic relation structures?

To answer those questions, data must first be preprocessed to be able to apply word-embedding methods on entity set expansion and relation extraction. For the preprocessing pipeline two different pipelines are tested: Jet, which is already implemented, and Stanford CoreNLP Natural Language Processing toolkit, which is available for multiple languages including English and German.

For the first sub-proposal a new entity set indexer is implemented, which returns a high-dimensional, dense vector for every word. To evaluate the indexer, a preprocessed corpus is used to extract similar entities to a given seed set. In several iterations the indexer proposes the possible entities, that are similar to the seed set. By simulating a semi-supervised learning, the precision of the system is retrieved for each iteration and in total. The evaluation is done for the original indexer, the newly implemented indexer in two variants and embedded vectors available by Google.

For the second sub-proposal, which is the relation extraction, dependency paths are extracted from the corpus. Then a seed relation is given to the system and the system is trained on the corpus to find similar relations. After this is completed, the acquired model is used to extract dependencies from a pre-annotated test set and it is evaluated, if the model detects the matching relations. This is done in a unsupervised approach.

To accomplish these tasks the thesis is divided in the following chapters: First related work is presented. As next, information about the used data is given. Then the entity set expansion method is described and evaluated. After that the relation extraction is described and evaluated. Following that, the workflow of the implemented system is laid out, including details about the multilingual approach, and the differences of the new implementations to the original system are compared. In the end, a conclusion of the master thesis is given.

4 Related Work

4.1 Overview of relevant methods

To analyze the text the process will be divided in three stages. In the first stage, the text is preprocessed with methods including tokenizing, sentence splitting, part-of-speech (POS)-tagging, named-entity recognition, and dependency parsing. For this task Java Extraction Tool (Jet) and Stanford CoreNLP Natural Language Processing Toolkit (Stanford CoreNLP) are used. In the second stage, entity set expansion is applied. For entity set expansion, a closer look at some of the recent research is taken. For example how to do fine-grained entity set refinement with user feedback, how to improve efficiency and accuracy in multilingual entity extraction, and how to represent words in vector space to be able to do calculations more efficiently and precisely. The third stage will deal with relation extraction. For this stage, the bootstrapping method 'Snowball' and a method for relation extraction on unlabeled data with distant supervision are explained.

Together these three stages equivocate to the new implemented information extraction system. As errors on one stage affect the accuracy of the next stages, it is important to achieve a high accuracy and low error rate on each stage.

4.2 Entity Set Expansion

Entity set expansion is the basis for a number of downstream information extraction tasks, such as relation and event extraction. As the entity-context relation is extracted in the entity indexer to build up the word-embeddings and these word-embeddings are used in one idea of the relation extractor, it is crucial to achieve a high precision in entity set expansion. Consequently, all errors in entity extraction will carry on in relation extraction and affect the success rate of relation extraction.

4.2.1 Fine-grained Entity Set Refinement with User Feedback

One method of entity expansion is developed by B. Min and R. Grishman. They propose an algorithm using relevance feedback, which includes positive and negative user feedback. The positive feedback is used to expand the feature set of an entity set, while negative feedback influences the feature weight. To find the positive entities, the user selects the first positive instance of the top n instances presented to him. For negative feedback, they create two ranking-based classifiers, which are constructed on two randomly split views of the feature space. Then they evaluate the results of the classifiers for instances. To get a list of negative instances, a instance that is ranked higher than the golden set size, is marked as positive, and an instance that is lower, is marked as negative. For negative feedback, the instances are used, where one classifier marked it as negative and the other one as positive. These instances are more ambiguous and more likely to be negative. By choosing those instances, which possibly contain a high information level, they achieve a significant improvement in comparison to the baseline algorith. This algorithm undergoes the instances that are most similar to the instance set centroid.[MG11]

4.2.2 Improving Efficiency and Accuracy in Multilingual Entity Extraction

Another approach for entity extraction was done by J. Daiber et al. They used DBpedia Spotlight as their IES. This system only needed tokenization as a language dependent processing step. For phrase spotting they first create candidates for possible annotations and then select the best candidates out of those. A ranking of candidates is computed based on a linear combination of features. For disambiguation they used a generative probabilistic model and for indexing they collected information from Wikipedia. All in all, they created a lightweight, multilingual NLP system.[DJHM13]

4.2.3 Learning Surface Text Patterns for a Question Answering System

Another approach where the Internet is used to train the model is presented by D. Ravichandran and E. Hovy in [RH02]. They investigated an approach to automatically learn regular expression by using bootstrapping on a seed set of question-answering pairs. They assume, that only simple questions are asked and search for patterns in the relation structure of the questions. They search the corpus for the question and answer term and only extract those sentences, which contain both. Then a suffix tree is built and only the phrases/patterns, which again contain both question and answer

term. These phrases are saved and the question and answer term are replaced by a placeholder each, so that all phrases saved have the same placeholder for the question terms and the same placeholder for the answer terms. Then they calculate the precision of the extracted patterns. Therefore they search the whole corpus for occurrences of the question word and extract the sentences, in which it appears. In these sentences they check which patterns are fulfilled. For fulfilled patterns, they check if the answer word matches the answer word of the pattern. For each pattern it can be calculated how often they are found and match the correct answer word and how often there is another word as answer. The result is a precision for each pattern. The patterns with the placeholders instead of the correct answer word can be used to find new questions and answers. This is done by using question patterns to find new questions, using the question terms and patterns to find answers, and score them again with the mentioned precision method. The results of this method are improved precision scores in comparison to a previous approach, which was based on a local corpus instead of the Web.[RH02]

4.2.4 Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations

The purpose of this paper is to research a weakly-supervised, general-purpose, and accurate algorithm to extract semantic relations.[PP06] Their goal is an algorithm with high performance, minimal supervision, no restriction on a specific domain and no restriction of the type of relations. It uses a bootstrapping method starting on a small set of seeds to iteratively extend the set and learn new relations/patterns. To score a patterns reliability, they present a new function and don't use the frequency, which is a common approach. Their function for calculating the reliability of a pattern p is a combination of the number of input instances, that are extracted by the pattern p, and the reliability of the input instances. The reliability of the input instances is associated of the seed input instances. The manually added input instances have a reliability of 1. All other input instances have a reliability between 0 and 1, depending how many other input instances they are associated with. To be able to use generic patterns, which are known to be high in recall, but low in precision, P. Pantel and M. Pennacchiotti are using the Web to avoid the low precision. Their assumption is, that in general in the Web correct instances of generic patterns will be instantiated by many reliable patterns. They also assume, that a correct instance will at least occur once in a reliable pattern even though it has a low recall.

4.2.5 context2vec:Learning Generic Context Embedding with Bidirectional LSTM

In this paper O. Melamud et al. present an approach to create vectors for whole contexts instead of single words. They use an unsupervised method to get generic context embedding of wide sentential context. The method learning algorithm is based on a recurrent neural network language model (NNLM). Their approach is a modification of Mikolovs' 'word2vec' model using continuous Bag-of-Word (CBOW). They save the context of a sentence in both direction. These resulting vectors are concatenated and then fed into another multi-layer perceptron. By that the resulting model can represent non-trivial dependencies. In comparison to that, 'word2vec' only considers the words in a window around the word as context. [MGD16]

4.2.6 A Neural Probabilistic Language Model

In this paper Y. Bengio et al. propose a probabilistic feedforward NNLM. They research options to estimate probabilities for yet unseen options of word combinations in a text. Therefore they calculate a word vector - they call it word feature vector - for every word in the vocabulary. To initialize these vectors semantic knowledge can be used, or they can be initialized with random values, or all are initialized with the same values. The first option will make the learning process for the system easier, but also is expensive as all semantic features have to be annotated manually. Then they set up an initial joint probability function for combination of words using the calculated word vectors. This function is a probability function of how high the chances are for one word to follow a specific word. It is a maximum likelihood estimator. To learn the word feature vectors a neural network is used. For the neural network a training set is required. The training set consists of words $w_1, w_2, ..., w_T$ with $w_k \in V$, where V is the vocabulary of a large, but finite set of words with $|V| >> |T|$. The goal of the neural network is to learn a function f to maximize the joint probability of sequences of words. Using a n-gram model then the following holds: $\hat{P}(w_t|w_1^{t-1}) \approx \hat{P}(w_t|w_{t-n+1}^{t-1}$. The function f can be represented by $f(w_t, ..., w_{t+n-1}) = \hat{P}(w_t|w_1^{t-1})$. For f applies: $\sum_{i=1}^{|V|} f(i, w_t, ..., w_{t+n-1}) = 1$, where $f(i, w_t, ..., w_{t+n-1})$ is the joint probability of word at position i in V in combination with sequence $w_t, ..., w_{t+n-1}$. This function can be split in two parts:

- A mapping function C, which provides a vector $C(i)$ for each word $i \in V$.
- A probability function g over the words in a sequence, and by that, a function over the vectors provided by the mapping function C: $g(C(w_{t-1}, ..., C(w_{t-n+1}))$

The output of g is a vector of length V. In this vector the i-th position estimates the probability $\hat{P}(w_t = i|w_1^{t-1})$. Therefore $f(i, w_t, ..., w_{t+n-1}) = g(i, C(w_{t-1}, ..., C(w_{t-n+1}))$

$C(i)$ is just the feature vectors, which are initialized somehow. g has to be either realized by another neural network or a parameterized function. The set of parameters from C and g has to be optimized to maximize the likelihood of function f. To ensure a positive probability for all positions a softmax output layer can be used. A softmax function maps a vector of dimension k to another vector of dimension k with all entries being in range $[0; 1]$ and summing up to 1.[BDVJ03]

4.2.7 Recurrent neural network based language model

In this publication a comparison of recurrent and feedforward NNLMs is given. While feedforward NNLMs only take the context of N - 1 words as history, recurrent NNLMs have an unlimited history by using neurons with recurrent connections. Also recurrent neural networks can use short term memory to deal with position invariance, which is not possible with feedforward NNLMs. While the feedforward model has a straight forward way through the layers, the reccurent models' input layer is a concatenation of current word/entity and the output from the previous layer.[MKB+10]

4.2.8 Scaling Learning Algorithms towards AI

In this paper Y. Bengio and Y LeCun compare shallow and deep learning models. They then propose a deep learning model and explain its advantages and disadvantages. Shallow models are characterized by having only one layer of fixed kernel functions. These kernel functions map the input by knowledge of the training set to the output. There are three different kinds of shallow architectures. Either the first layer is a fixed basis function to predict the output or a kernel machine architecture, in which the preprocessing is a vector of values related to the learning samples, or a set of simple functions derived by learning under supervised training. Deep architectures on the other hand are multiple-layer models of parameterized non-linear modules. Each layer has to be trained. Deep architectures allow a wide range of functions in a more compact way than shallow architectures. The advantage of deep architectures is, that multiple calculations can be done in parallel with multiple functions. They also claim, that most functions, which can be represented compactly by a deep architecture, can't be represented in a compact way by a shallow architecture. [BL+07]

4.2.9 Efficient Estimation of Word Representation in Vector Space

Tomas Mikolov et al. present a method for word representation. It is based on neural language models, but doesn't have the disadvantage of the complexity caused by the non-linear hidden layer. They introduce two architectures:

The first one is similar to feedforward NNLM. Instead of the hidden layer, they use a projection layer which is shared for all words. This kind of model is called CBOW model. More specifically the model used is a CBOW, which uses continuously distributed representation of the context. For the bag-of-words model, the words in a specified range k around the word, which shall be estimated, are taken into consideration. That means, that the k words before the word and the k words after the word are used to calculate a vector. The model is called bag-of-words, because the order of words has no influence on the vector.

The second architecture is the continuous skip-gram model. It uses a log-linear classifier with continuous protection layer and predicts words before and after the current word within a certain range. Increasing the range improves the results, but also increases the computational complexity. They also concluded that more distant words usually contain less important relation information for the current word. Therefore more distant words are weighted less.

The result of this paper is to demonstrate that with simple model architectures it is possible to compute accurate high dimensional word vectors from large data sets due to the low computational complexity. [MCCD13]

4.2.10 word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method

In this paper the negative sampling used in 'word2vec' by T. Mikolov is explained. One important basis is the skip-gram model. The goal of the skip-gram model is to create a vector for a word representing it in the context of the corpus. Therefore the probability $p(c, w; \theta)$ of a word w appearing in the context c for the parameters θ is maximized: $arg\ \underset{\theta}{max} \prod_{w \in Text} [\prod_{c \in C(w)} p(c \mid w; \theta)]$, with D as the set of all word and context pairs from the text.
To extract vectors for the words, the skip-gram model is used to calculate probabilities for a context word appearing close to the word. The neural network builds those vectors on the basis of word and context pairs. Depending on the number of occurrences of a pair, the probabilities are computed.

In the negative sampling approach, which is similar to the skip-gram model, they don't model $p(c|w)$, but a "quantity related to the joint distribution of w and c."[GL14] Their model works the following way: First of all they set up a probability if a word-context pair is from the corpus:
$p(D = 1|w, c)$
If it is not from the corpus, then the probability is:
$p(D = 0|w, c) = 1 - p(D = 1|w, c)$
As for the skip-gram model th distribution is controlled by parameters θ:
$p(D = 1|w, c; \theta)$
And as before, the goal is to maximize the parameters θ that all pairs came from the corpus:
$arg\ \underset{\theta}{max} \prod_{(w,c)\in D} p(D = 1|w, c; \theta)$
Because taking the log does not alter the results and sums are faster to calculate than products, the equation is transformed to:
$arg\ \underset{\theta}{max} \sum_{(w,c)\in D} log\ p(D = 1|w, c; \theta)$
As the probabilities for any of the pairs should be in range $[0, 1]$, the softmax function can be used:
$p(D = 1|w, c; \theta) = \frac{1}{1+e^{-v_c * v_w}}$, with v_c and v_w being the vectors for context c and word w
This leads to the equation:
$arg\ \underset{\theta}{max} \prod_{(w,c)\in D} \frac{1}{1+e^{-v_c * v_w}}$
The problem with this optimization is, that it reaches a maximum if all vectors have the same values. Therefore negative sampling is used. This means, that a set D' of pairs (w,c), which are not available in the corpus, is introduced. These pairs must have a low probability. By that the optimization problem changes to:
$arg\ \underset{\theta}{max} \prod_{(w,c)\in D} p(D = 1|w, c; \theta) \prod_{(w,c)\in D'} p(D = 0|w, c; \theta)$
$= arg\ \underset{\theta}{max} \sum_{(w,c)\in D} log \frac{1}{1+e^{-v_c * v_w}} + \sum_{(w,c)\in D'} log \frac{1}{1+e^{v_c * v_w}}$
The success of this method in producing good word vectors for predicting similarities is correlated to the Distributional Hypothesis by Z. Harris (1954), that states, that words, which appear in similar contexts, have similar meanings. [GL14]

4.2.11 Web-scale distributional similarity and entity set expansion

Another application of entity set expansion is proposed by P. Pantel et al. Their focus is on applying entity set expansion and calculating similarities between entities on the web. Therefore they have a seed set and calculate a centroid. They then rank all entities according to their similarity to the centroid and apply a cutoff to the ranked list. As similarity functions they use standard similarity functions like Cosine, Jaccard, Dice and Overlap. They create a gold standard of different entity sets and apply a list of entities

to each of the sets. They test their method by creating multiple copies of the sets in the gold standard and test the proposals for different seed set sizes. Their goal was to check, whether the system would recognize the same sets as in the gold standard.[PCB+09]

4.3 Relation Extraction

4.3.1 Snowball: Extracting Relations from Large Plain-Text Collections

One of the more commonly used ideas in semi-supervised relation extraction is bootstrapping, e.g. the 'Snowball' method. The Snowball method starts in seed tuples, which are the source-target pairs of a relation ship. The method finds occurrences in the documents, where these tuples exist close to each other and analyzes the surrounding structures to create patterns, by tagging the text between the two entities of the tuple. After pattern creation, the systems looks for more occurrences of the new patterns. Each tuple is then rated with a degree of match to a pattern and is assigned the one with the highest degree. This process of rating together with information about the selectivity of the patterns is used to decide which tuples will be used as seeds from then on. One possible ending criteria is the missing of new seeds produced in an iteration. [AG00]

4.3.2 Semi-Supervised Semantic Pattern Discovery with Guidance from Unsupervised Pattern Clusters

In natural language processing the lack of labeled data for new domains or tasks is a common problem. The approach of A. Sun and R. Grishman deals with that problem using the Distributional Hypothesis by Z. Harris (1954) and extended it to also cover patterns. They claim that patterns that the patterns that occur within a similar context tend to have similar meaning. To do the preprocessing, they use Jet and the dependency parser of Stanford CoreNLP. After retrieving the dependency paths, all paths are clustered based on distributional similarities. Within these paths each pattern is assigned to a cluster, thence making it possible to calculate a score dependent on how strong a named-entity (NE) pair is associated with a cluster. The first step of bootstrapping is to use the seed patterns to match new NE pairs and evaluate NE pairs. The NE pairs are then used to search for new patterns and those patterns are ranked once more. Pairs are evaluated by calculating the harmonic mean of two confidence scores. On the one hand "the confidence of its association with the target cluster as well as the confidence given by the patterns matching it." [SG10] By this evaluation technique they claim to achieve better rankings than with the standard bootstrapping. After ranking all patterns the K

top ranked ones get accepted, the system then either loops to find more patterns or stops. The stop criterion of this system is a trigger by the semantic drift, if the system tries to accept a pattern, which does not belong to the target cluster, the system stops.[SG10]

4.3.3 Distant supervision for relation extraction without labeled data

In this paper the main idea is to use distant supervision instead of supervised or unsupervised methods. Distant supervision assumes, that if a relation exists between two entities according to a knowledge base and the two entities appear again in another sentence, that it will be the same relation between them again. In this paper a database was used as supervisor. This eliminates the problem of domain-dependence. The system of this paper is also capable of matching information from different mentions of the same relation.[MBSJ09]

4.3.4 Distinguishing antonyms and synonyms in a pattern-based neural network

K.A. Nguyen et al. propose the hypothesis that antonymous words appear more often in the same sentence as synonymous words. They make use of the dependency paths of the sentences. For each node in the dependency tree they save the POS tag, the lemma, the dependency label and the distance label. For each of these features they calculate word-embeddings based on the extracted patterns of a large-scale web corpus using a recurrent NNLM. The vector of the node is the concatenation of all feature vectors. To distinguish between synonyms and antonyms Nguyen et al. present two models.

In the first model they give all patterns for a word pair (x,y), where a patterns represents a path from x to y, to a recurrent NNLM with long short-term memory (LSTM). This results in the vector v_p of the pattern p. The vector for the word pair (x,y) is the sum over all patterns $P(x, y)$ corresponding to the pair (x,y) divided by the sum over the frequencies c_p of the patterns: $\vec{v}_{xy} = \frac{\sum_{p \in P(x,y)} \vec{v}_p * c_p}{\sum_{p \in P(x,y)} c_p}$ This vector v_{xy} is given to a logistic regression model to predict if the pair of the vector are synonyms or antonyms.

In the second model they use additional to the patterns also the distribution of the target pair to calculate the word-embeddings. Therefore the vector $\vec{v}_{comb(x,y)}$ for the pair (x,y) is the concatenation of the word-embeddings of x and y $\vec{v}_x$ and $\vec{v}_y$ and the patterns vector $\vec{v}_{x,y}$. This vector $\vec{v}_{comb(x,y)}$ is again the input for a logistic regression model to predict if the the word pairs are synonyms or antonyms.[NWV17]

4.3.5 Improving hypernymy detection with an integrated path-based and distributional method

In this paper V. Shwartz et al. present an approach to detect hypernymy. This approach uses both path-based as well as distributional methods in an integrated approach to predict these kind of relation. They also use a recurrent NNLM with LSTM to calculate the embeddings and use a tree structure to present the dependencies. The path consists of the edges . For each of the edges the NNLM learns a vector. The vector is a concatenation of the vectors for the lemma, the POS tag, the dependency label and its dependency direction. In order to get a vector $\vec{v}_p$ for the path from x to y, the vectors for the edges are used as the input for an LSTM encoder. To represent both the path-based and distributional method, the word-embeddings of x and y are concatenated with the vector $\vec{v}_p$ of the path. Therefore the vector for the word pair (x,y) is $\vec{v}_{xy} = \vec{v}_x \oplus \vec{v}_{(x,y)} \oplus \vec{v}_y$.[SGD16]

4.4 Framework ICE

The framework used in this master thesis is Integrated Customization Environment (ICE). ICE extracts entities and relations, which occur between those entities specified by the user.[HG15]

ICE divides up into 5 major steps.

- Preprocessing
- Key Phrase extraction
- Entity set construction
- Dependency paths extraction
- Relation pattern bootstrapping

For the preprocessing step the JET NLP pipeline was used. This pipeline was developed for the English language and is based on the Penn Treebank POS tags.[1] After preprocessing the documents, the user wants to extract information from, the system guides the users to define entity and relation types of their interest. The idea of ICE is to create a system, which a user, who has no background in computational linguistics, can use. Therefore ICE only needs a few examples of an entity or relation, set by the user, and then calculates possible entities, belonging to that entity set. The user then

[1] https://cs.nyu.edu/grishman/jet/guide/PennPOS.html

judges, which of the top ranked entities are correct and which aren't. By that the user doesn't have to give a comprehensive list of a entity set. The same is done for relation sets.[GH14] In order to get a high quality relation extraction, all previous steps have to have a low error rate as well, as errors will propagate through the different working steps.

5 Data

To be able to compare the newly implemented methods with the existing method, a semi-supervised evaluation method is used. In order to do so a corpus and an associated gold standard is needed. As corpus 6391 web news posts from U.S. Drug Enforcement Administration (DEA)[1], which was already collected by the New York University (NYU), are used. These texts sum up to a total of 33.1 MB.

As shown in the example 10.3, each post contains of a date, a contact, a number, a title of the document and a description of the court case. From these news posts the entities and dependencies are extracted in the preprocessing step.

The gold standard for entity set expansion was originally created by Prof. Ralph Grishman and Dr. Yifan He. During this thesis the embedded methods proposed additional correct proposals for drugs, which didn't exist in the gold standard yet. These are included in the gold standard in order to give a precise comparison of the performance of the systems. The entire gold standard originally had 120 drug entities. After adding the correct proposals by the implemented methods, the number of drug entities in the gold standard went up to 223. This also has the advantage, that the results of the evaluation is more trustworthy. In the original system from the total of 200 proposals only 120 proposals could possibly be correct. Therefore the precision, which is the number of correct proposals divided by the number of total proposals, could never be 100%. With 223 entities in the gold standard it is possible to achieve a precision of 100%.

Each entry in the gold standard is a pair of the name of a drug and a weight, which determines how much the drug name should be weighted when calculating the centroid of the seeds. This weight can be used to tell the system which drugs are preferred and more important. In this master thesis the entities in the gold set are weighted by the frequency of their appearance in the corpus. In the gold standard also combinations like 'liquid methamphetamine' , 'drug methamphetamine' or 'pure heroin' are included as they are a subcategory of the drugs 'mephamphetamine' or 'heroin'. As these subcategories are recognizes as their own entities, it makes sense to include them in the gold standard. Also the set of contrary entities of 'liquid' and 'pure' is limited. Therefore all combinations

[1] http://www.justice.gov/dea/index.shtml

can be covered. In example, the contrary of 'pure' is 'impure', the same holds for 'liquid', it is a state and other possibilities are only 'solid' and 'gas'. On the other hand entities like 'good heroine' are not taken to the drug corpus. The number of combinations of adjectives and drug names is too large to cover all of them.

For relation extraction also an already existing gold standard is used. This gold standard consists of 25 XML-files, which sum up to to a total of 172KB. Each file has a similar structure to the example shown in 10.4. First the text is given in the same structure as in the DEA drug corpus. The texts from all 25 files are the test set for relation extraction. Then the relations appearing in the text are listed. This can be a single one as in the example 10.4 or multiple relations, or also zero relations, if there are non existing for 'sell'. If there are relations, then the subject and object are listed for each relations as 'arg1' and 'arg2'. For each relation, subject, and object the start and end position in the text is given. The relation, subject, and object represent the gold standard. These are the relations, which should be found by the relation extraction method.

6 Methods

In this chapter, the actual methods of this master thesis are presented. While the focus of this thesis is on the research of the entity set expansion and also looks at the relation extraction, the preprocessing of the corpus is an important factor for the performance of the research tasks. Therefore the preprocessing will be elaborated as well.

6.1 Preprocessing

As Jet is used for the preprocessing of the entity set expansion and relation extraction of the original system, it is used also for the newly implemented methods to be able to have a direct comparison. Therefore, the preprocessing steps for Jet are explained in this chapter. Before entity set expansion or relation extraction can be applied, entities and relations of the training corpus have to be found.
Preprocessing splits up into 4 different types of annotations:

1. tokenizing
2. sentence splitting
3. POS tagging
4. parsing

A tokenizer annotates the tokens of a text. The Jet tokenizer distinguishes between three different types of tokens:

1. words
2. numbers
3. special characters

After these types are annotated, the corpus can be split into sentences. This is done by defining spans of the corpus as a sentence. I.e. a question mark or exclamation mark marks always the end of a sentence. For a period more special cases have to be distinguished.
In order to extract dependencies from the sentences in the corpus, the POS tags of the tokens has to be known. To get these POS tags, a lexicon lookup is used. Each token is looked up in the lexicon and annotated with the possible annotations. This means, that tokens with multiple possible annotations are marked with all of them. The correct annotation has to be found by the POS tagger. This tagger ranks all possible annotations according to their frequency and also according to how likely they appear with surrounding tokens. The tagger then picks the annotation, which is the most likely for this sentence.
After tagging each token with its POS tag, dependencies can be annotated. The goal is to have a dependency structure for each sentence. Therefore syntactic relations between the entities of a sentence are extracted. The precision is highly related to the complexity of the grammar of the language, in which the corpus is written. For English, dependency parsing is fairly easy. Other languages, which have more freedom in the structuring of sentences, are more difficult to parse. I.e. in English the place is always written before the time: 'The child arrived at home at 8pm.' In order languages the phrase could also be written as 'The child arrived at 8pm at home.' The more distinct a grammar is, the easier is it to achieve a high precision of the dependency parsing. But also for English there are sentences, where the structure is not directly obvious, but has multiple options.[Gri] Therefore an easy-first dependency parser is used:

6.1.1 An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing

The basic idea of this algorithm is to start with the relations, which are easy to detect and then use those already annotated relations to narrow down the range of possibilities for the remaining parts of the sentence.
A bottom-up parser is used to connect always two constituents of a sentence. The parser is initialized with a list of partial entities $p_1, ..., p_k$, called pending, and the n words of the sentence $w_1, ..., w_n$. In each iteration one of two neighboring entities is set as the parent and the other one as the child.
After n-1 iterations a complete parse tree is created. In order to set the easy relations first, they use a learning function and a feature representation to score the pending entities. The learning function is trained on an annotated corpus. That way the learning function can be trained, that it annotates every sentence in the corpus correctly. Every time the parser proposes an invalid option, the weights for the current highest scoring

valid action are increased and the weights for the invalid action are decreased.
The feature representation is used to choose desired actions over less desired actions, but also in case there are multiple desired actions for a pending entity to wait until more information is available for this entity and the choice between the desired action is more distinct.
For the features the POS tags of the partial entity and the surrounding partial entities in a specified range and the length of the entities are taken into consideration.[GE10]

With the entities and the relations between the entities of a sentence in the corpus annotated, the foundation is set for conducting entity set expansion and relation extraction.

6.2 Entity Set Expansion

In this chapter the techniques used for entity set expansion are explained.

In the preexisting standard approach of ICE a list of nouns are extracted from the corpus and for each noun a list of entities, found in the context of the noun, are saved. For each pair of noun and context entity a similarity value is calculated using cosine similarity.

While this approach worked accurately, the new approach is to calculate a multi-dimensional vector for each noun in context to the other words in the dictionary. This approach is based on the 'Distributional Hypothesis' by H. Zellig [Har54], claims, that similar words appear in the same context more frequently than words, which are dissimilar. To calculate the values of the vector's dimensions a 2-layer neural network called Word2Vec is used. The Word2Vec model takes a text's corpus as input and gives a vector for each word as an output. The vector represents the word in respect to its context. The algorithms are always tested against a gold standard of a drug words set; here the vectors are extracted from a corpus containing charges on drug crimes. The training texts for the corpus are taken from the DEA[1].

For training the model, on the retrieved terms of the corpus, the first attempt was to use the Word2Vec model provided by deeplearning4j for java.[Tea] This model had the problem, that it internally grouped words together to form entities, which didn't overlap with the grouping performed in the preprocessing step. This led to the problem, that there were no vectors for some words in the vocabulary. For these words it wouldn't be possible to calculate similarities. Therefore a new approach was needed.

[1] https://www.dea.gov/index.shtml

The next approach was to use the original 'Word2Vec' implementation by Mikolov et al.[MCCD13] It gets the corpus split into one sentence per line as an input and returns vectors for single words only. The vectors for multi-word terms can be created by adding up the vectors of the single words. The assumption for this approach is, that the addition of the vectors of words, which represent the meaning of a word in its context, will represent the true meaning of the sequence of the words.

For the 'Word2vec' method by Mikolov the continuous skip-gram model is used to calculate the vectors as presented in 4.2.9. By using Mikolovs' word-embeddings, the goal is to create a model, which can be easily extended, if new texts are given and which predicts similar entities to a seed entity with a higher precision for low iterations. Low iterations are necessary to minimize human interaction, as this is expensive and slows the progress down, when the system has to wait for human input.
But this approach has also disadvantages to the original method. While in the original method all similarities were calculated during indexing, Mikolovs' word-embeddings are vectors, which represent a word in its context. Similarity has to be measured during runtime of the algorithm. This can be expensive, as it is in O(d) with d being the dimension of the vectors. For the sparse vectors of the original approach only one position has to be read. Similarity is already calculated during the indexing step.
A disadvantage of adding the vectors of single terms to retrieve the vectors for multi-word terms is, that if the single words are used in different contexts than the combination of both, it might be inaccurate to just add them up. In example, when comparing the two entities 'cocaine dealer' and 'cocaine'. As 'cocaine dealer' is the addition of the vectors 'cocaine' and 'dealer', the similarity between 'cocaine' and 'cocaine dealer' might be high. But this isn't wanted, because 'cocaine dealer' is a person and 'cocaine' is a drug.

Therefore, in the next attempt all words in a multi-word term are combined by an underscore and then run the 'Word2Vec' algorithm on the adjusted corpus. To achieve the best distinction of terms in their context, the replacement of words is started on the longest multi-word terms. If not started on the longest multi-word terms first, then it might happen, that a term consisting of two words is replaced in the corpus, but if it is a subset of another term, this longer term won't be recognized anymore cause the corpus now contains underscores on those terms.

To avoid the differences between the Jet parser and the parser used by 'Word2Vec', the final attempt is to use the 'Word2Vecf' algorithm by Omer Levy and Yoav Goldberg.[LG14] This algorithm is based ob the 'Word2Vec' by Mikolov et al., but instead of taking the sentences as input, it uses a list of any word plus context word, a file with the counted words and a file with the counted context words. These files are created by Jet and therefore overlaps completely with the results of the preprocessing. Another advantage of this modification of the original 'Word2Vec' is, that 'Word2Vecf' can perform multiple

iterations on the data. This increase the accuracy of the vectors by eliminating mis-predictions.

6.3 Similarity functions

After all words are represented by multi-dimensional vectors, the similarity function determines how similar two vectors/terms are in respect to the context they were trained on. Choosing a fitting similarity function is crucial for the precision of the predictions. The similarity function decides if two entities are similar and can be added to the same entity set or not. In this thesis three standard similarity functions are applied: Cosine, Jaccard and k-nearest neighbor (kNN). Also four custom similarity functions are tested for both Cosine similarity as well as kNN to get more precise suggestions by the IES.

6.3.1 Cosine Similarity

Cosine similarity is a widely used similarity function. It is defined as the dot product of two vectors divided by the product of their lengths, which is the angle between two vectors. Because it is assumed that similar words will have similar vectors, or that the vectors point towards a similar direction, but have different lengths, Cosine Similarity is usually a popular choice.
Cosine similarity normalizes the vectors and therefore ignores the magnitude of the vectors.

$$cos(\theta) = \frac{termVec * centroidVec}{\|termVec\| * \|centroidVec\|} \tag{6.1}$$

The problem of cosine similarity is, that a multi-word term is similar to the words in the multi-word term. In reality those words are often related, but not necessarily similar. An example is 'cocaine', 'cocaine powder' and 'cocaine supplier'. While 'cocaine' and 'cocaine powder' have similar meanings 'cocaine supplier' shouldn't be similar to 'cocaine', but rather to other drug suppliers. In cosine similarity comparisons between all three examples will produce high similarity as a result.

6.3.2 Jaccard similarity

A similarity function, which is a good counter to the similarity problem of multi-word terms, is a modification of Jaccard similarity. For Jaccard similarity, a multi-word term is split into its consisting words and then the centroid, which consists of all seed words, is checked if it includes any of the term words. Jaccard similarity then divides the number

of items in the intersection of the centroid and the term by the number of items in the centroid and the term minus the number of intersecting items.

$$\begin{aligned} jac(term, centroid) &= \frac{\|termVec \cap centroidVec\|}{\|termVec \cup centroidVec\|} \\ &= \frac{\|termVec \cap centroidVec\|}{\|termVec\| + \|centroidVec\| - \|termVec \cap centroidVec\|} \end{aligned} \tag{6.2}$$

In case there aren't any intersecting words, which is also the case for new single-word terms, cosine similarity is computed to measure the similarity.

6.3.3 k-nearest neighbor similarity

Another similarity function, applicable to vectors, is the kNN classifier. This classifier calculates the distance of one vector to the surrounding k vectors, which already are appointed to a specified class. The vector gets assigned the class, which has the most other vectors in the set of surrounding ones.

6.3.4 Penalty similarity functions

After introducing the standard similarity function, the custom optimizations to the standard similarity functions, that are invented for the specific use case investigated in this thesis, are introduced. All penalty functions introduced in this section are implemented both for cosine similarity as well as kNN similarity.
As there is no differences to the functionality if they are applied with the cosine of kNN method, they are only described for cosine similarity. In this thesis kNN is only tested in the original version and with the penalty similarity function as this similarity produced the best results for cosine similarity.

Cosine similarity function with eliminating existing words

In this thesis the first custom similarity function was based on the idea of eliminating the intersecting words and calculating the cosine similarity on the remaining centroid and term words. If the centroid and the multi-word term have intersecting sets of words, then the vectors of the intersecting words are subtracted from both the centroid and the multi-word term. The result is a temporary centroid and a temporary term. The idea is

based on the assumption, that, if a term is similar to the centroid, each single word of the term is similar to the centroid as well.

$$\begin{aligned} termExVec &= termVec - (termVec \cup centroidVec) \\ cosEx(\theta) &= \frac{termExVec * centroidVec}{\|termExVec\| * \|centroidVec\|} \end{aligned} \tag{6.3}$$

The elimination of existing words counters the disadvantage of cosine similarity of accepting multi-word terms being similar to the centroid, when one word of the multi-word term already exists in the seed set. On the other hand, if a multi-word term consists of two drug names, e.g. 'crack cocaine' or if it is a subgroup of a drug like 'crystal meth' of 'meth', then the elimination would prevent these words from being added to the seed set. Therefore the next idea is not to eliminate them completely, but to penalize multi-word terms with a factor δ.

Existing word penalty function

This function is similar to the cosine similarity function of eliminating existing words. Other than eliminating the intersecting words, the centroid is not adjusted at all and the intersecting words in the multi-word term are just penalized with a factor instead of removing them completely. This means, the words in the multi-word term, which don't appear in the seed set are not penalized at all, only the existing words get penalized. By only penalizing with a factor δ, there is a higher ranking for single words, as well as, multi words, which do not not have any existing words in the seed set. The idea behind this function is, that most drugs only consist of a single word or are a combination of a single-word drug and another word. That way single-word drugs will be the top ranked entities of this penalty function.

$$\begin{aligned} termExVec &= termVec - \delta * (termVec \cup centroidVec) \\ cosEx(\theta) &= \frac{termExVec * centroidVec}{\|termExVec\| * \|centroidVec\|} \end{aligned} \tag{6.4}$$

But one side unwanted side-effect is the fact, that multi-word entities, which don't have any word in common with the seed set, are not penalized and might be ranked higher than the multi-word entities with an existing seed word.

To avoid this, another similarity function is useful:

Multi-word penalty function

This similarity function penalizes all multi-word terms. The algorithm is tested on the drug data set, in which most drugs are either single words, abbreviations, which are also

represented as a single word, or multi-words consisting of a single-word drug. Therefore the idea is, that most multi-word terms, which are likely to be similar to the centroid, are consisting of a drug plus another word, e.g. 'cocaine powder'. These multi-word terms won't be chosen in the first iteration due to the penalization. Presumably, as soon as the drug name included in the multi-word term is added to the seeds, the multi-word term will move up in the ranking and then be chosen as a possible candidate.

$$\begin{aligned} if\ term\ consists\ of\ multiple\ words:\ similarity &= \delta * cos(\theta) \\ else:\ similarity &= cos(\theta) \\ cos(\theta) &= \frac{termVec * centroidVec}{\|termVec\| * \|centroidVec\|} \end{aligned} \tag{6.5}$$

By penalizing all multi-word entities, the ones, which have a word in common with the seed set, will still be ranked higher than the other multi-word entities.

Combined multi-word & existing word penalty function

In this similarity function the advantages of both the multi-word and the existing word penalty functions are taken and combined to a new similarity function. First, the term is checked to see if it consists of multiple words.
For single-word terms, the cosine similarity of the term and the centroid is directly computed: $cos(\theta) = \frac{termVec * centroidVec}{\|termVec\| * \|centroidVec\|}$

For multi-word terms, the similarity is calculated in the following way:
The centroid is checked if it contains words of the multi-word term.

1. if yes: penalize the intersecting words and calculate the similarity on the new term vector:

$$\begin{aligned} termExVec &= termVec - \delta * (termVec \cup centroidVec) \\ cosEx(\theta) &= \frac{termExVec * centroidVec}{\|termExVec\| * \|centroidVec\|} \end{aligned} \tag{6.6}$$

2. if not: penalize the whole term:

$$\begin{aligned} if\ term\ consists\ of\ multiple\ words:\ similarity &= \gamma * cos(\theta) \\ else:\ similarity &= cos(\theta) \\ cos(\theta) &= \frac{termVec * centroidVec}{\|termVec\| * \|centroidVec\|} \end{aligned} \tag{6.7}$$

With $\delta > \gamma$ this leads to the following ranking of terms. Similar single word terms are ranked the highest, then ranked below are the multi-word terms, which contain the terms of the seed set, are given and at last are the multi-word terms, which show no existing words in the seed set.

Cosine with eliminating adjectives

Another idea for a similarity function, which wasn't implemented, is to check every multi-word entity for a match of its single word with the seed set. If that is the case and the other words are adjectives, the entity can be accepted. An adjective does not alter the meaning of the drug itself, unless is negates it, e.g. 'fake drug'.
All other similarity functions potentially propose a entity like 'good heroine' and reject it, as it isn't included in the gold standard. A combination of a drug name and an adjective can't be included in the gold standard, because the number of combinations is too large to cover everything. On the other hand, analyzing a entity on the tags and eliminating adjectives is much easier to accomplish. This could potentially increase the precision of the system significantly.

6.4 Ranking and re-ranking entities

Using one of these similarity functions, similar entities to another entity can be easily detected by comparing them according to the set similarity function. The goal of the similarity computation is to build up entity sets.

This can be achieved in the following way:
To start an entity set, seeds have to be set. As these seeds are in the vocabulary, vectors exist for these seeds. Consequently, an average vector of all seed entities can be calculated. This average vector is called the centroid.
In the next step, all entities are compared to this centroid using the specified similarity function. As the result of all similarity functions is a value $p \in [0, 1]$, the entities can be ranked according to their similarity values in descending order.
From the ordered, ranked entities the top k entities are extracted. These k entities are the proposals and are evaluated manually by the user.

If the entities are correct, they are added to the seed set, otherwise, they are added to the negative seed set. The negative seed set is used to reduced the impact of the features, which are responsible for proposing the wrong entities as it is described in 4.2.10.
This sequence of ranking the entities, proposing and evaluating the top k entities and updating the positive and negative seed sets is repeated iteratively. Therefore in each

iteration the entities with the highest probability are predicted.
Recalling the Distributional Hypothesis by Z. Harris (1954) and that the vectors represent the entities by the context they are used in, a high probability, which correlates to a high similarity between two vectors, means, that the two entities compared, are similar. Consequently, the entities, which return a high probability by the similarity function, have the highest probability of being correct proposals.
By iterating and re-ranking the entities the system can learn faster than evaluating the whole entities on possible positive matches.

By using a semi-supervised approach to build up entity sets, in which in each iteration the user has to judge for 20 entities, whether they belong to a specific entity set or not, the model learns fast to judge, which entities are correct and which are wrong.

With the entity set expansion methods presented, the next step will be the relation extraction.

6.5 Relation Extraction

In this chapter the newly implemented relation extraction method is explained.
Opposing to the semi-supervised approach to predict new relations used in the original method in ICE, in this thesis a unsupervised approach is proposed. While it is easy for a human to decide whether an entity belongs to an entity set or not, it is not as unambiguous, whether two relations belong to the same relation set. Another advantage on an unsupervised method is, that it doesn't require human input, when working on a new set of relations. Therefore the following method is learning only during training time and is then evaluated on the test set without any changes to the model during testing. This thesis focuses on the extraction of relations centered around verbs. As the 'Distributional Hypothesis' is used for entity set expansion, a basic assumption for relation extraction is, that if relations of entities occur in similar ways, then the relations are similar. I.e., if two verbs are used in similar contexts like 'He sold cocaine' and 'He peddled cocaine', then they have a similar meaning.
For relation extraction also an iterative learning approach is used. In the first iteration a seed entity is given. In this thesis, this is the verb 'sell'. Then the algorithm scans the whole corpus for sentences, in which the entity 'sell' exists. In these sentences all syntactic relations with 'sell' as the source or target are extracted. A syntactic relation has the following important features:

- sourceWord
- sourcePos

- sourcePosn
- type
- targetWord
- targetPos
- targetPosn

I.e. when analyzing the sentence ’The indictment charges one count of conspiracy to distribute narcotics.’ One relation extracted is between the two words ’distribute’ and ’narcotics’.

- In the field ’sourceWord’ is the source entity of the relation, in this example it is ’distribute’.
- In the field ’sourcePos’ is the POS tag of the entity according to the annotation tags of the Penn Treebank saved. ’distribute’ is a verb, therefore the tag is ’VB’.
- In the field ’sourcePosn’ is the start position of the entity in document. ’273’ in this case.
- In the field ’type’ the relation type is saved. As ’narcotics’ is the direct object of ’distribute’, the relation is ’dobj’.
- In the field ’targetWord’ the entity ’narcotics’ is saved.
- In the field ’targetPos’ ’NNS’ is written, because ’narcotics’ is a noun in plural form.
- In the field ’targetPosn’ ’304’ is noted.

The syntactic relations are created as described in section 6.1.
When searching for similar relations of an entity, the first step is to search for sentences, in which the entity appears. In these sentences all relations with the entity as ’sourceWord’ or ’targetWord’ are saved.
One primitive approach to extract similar relations is to search the corpus for appearances of the exact same combination of different relations as it appears in one of the found sentences. After finding these appearances, extract the key entity, that is used in these new found sentences and repeat the process. This approach is based on the assumption, that similar entities or synonyms of an entity would appear in exactly the same context including the use of the prepositions and other grammatical constructions.
If a synonym is only used in different sentence structures, then it could never be matched by this approach. On the other hand, entities, which get matched, are very likely to be a synonym to the seed entity and that the relation has a similar meaning.

A more advanced approach is to first search for sentences with relations containing the seed entity. Then extract the subject and direct object of the verb in the sentence. The

result is for example a noun-verb-noun relation as in 'The drug dealer sold drugs'. From this relationship use the two nouns to search the corpus for occurrences of this pair and whether they have a noun-verb relation in common. The assumption is, that if entities occur in multiple relations, the relations have similar meanings. So any relation, which is found in the corpus by the described method, is likely to have a similar meaning. This method has the advantage, that it is less strict in accepting relations, as when only looking if certain key features are given in a sentence. This is also a disadvantage as it opens up to false positives.

To reduce the number of false positives this method is extended into a third approach to extract relations. In order to avoid false positives, the embedded vectors, which are already calculated in the entity set expansion, are used. These vectors are calculated using the actual entities in context rather than their POS tags.
In this approach, an initial sentence is used. From this initial sentence, the verb is extracted, added to the seed set, which is empty on initialization. Then appearances of the verbs, saved in the seed set, are searched in sentences in the corpus. From the sentences, which contain a verb from the seed set, the subject and object are extracted. These subject-object pairs are searched in the corpus, to find more sentences, in which they appear in close distance as subject and object of a verb. This results in noun-verb-noun pairs again.
From the sentences the verbs are extracted. The word vectors are used to compare the newly found verbs with the original verb of the initial sentence. They are ranked by their similarity calculated with cosine similarity on the word vectors saved in entity set extraction. The verbs, which provide a similarity value higher than k, with $0 \leq k \leq 1$, are added to the seed set.
Then the process is repeated. Sentences are searched with the verbs of the seed set appearing, from the sentences subject and direct object are extracted and appearances of these subject-object pairs are searched in order to find new verbs. One cutoff criteria could be to stop searching, if there are no new verbs found in x iterations. In the end there is a list of relations, which are similar to the initial relation. The relation consists of a subject and direct object of a verb. This list of relations can be used to check whether the model finds the relations manually tagged as correct in the test set and precision, recall and F1 score can be calculated.

One could even take this approach further and not only check if the verbs have similar embedded vectors, but check the verb, subject, and object for similar entities in the vocabulary, by comparing their word vectors and then check, if they appear in similar contexts. Or the embedded vectors of the subject and object could be used to set up entity sets and check if any combination of the two sets appear in the corpus. This could extend the recognition rate of the algorithm, but wasn't tested in the scope of this master thesis.

With the new approaches for entity set expansion and relation extraction proposed, the next step is to evaluate the algorithm and compare their results with the original implementation if ICE.

7 Evaluation

The evaluation of the proposals of this master thesis divides into two sections. First the proposed entity set expansion algorithm is evaluated and advantages and drawbacks are given, then the same is done for the proposed relation extraction method.

7.1 Evaluation Entity Set Expansion

7.1.1 Evaluation Method

For the evaluation of the implementation of entity set expansion a corpus is needed. As training corpus a collection of drug crimes of the DEA[1] collected by NYU[2] is used. The gold annotations were created by Prof. Ralph Grishman and Dr. Yifan He. The structure of the text in the corpus is described in Section 5. For the evaluation of the new implemented entity set expansion algorithm, a semi-supervised information extraction is simulated.
To achieve this, two seeds are fed to the algorithm and these set the centroid for the first iteration. In this evaluation the two seeds are 'methamphetamine' and 'oxycodone'. The simulation extracts the top 20 ranked entities by the entity set expansion algorithm and compares these proposals with the existing entities in the gold standard. If a proposal exists in the gold standard, it is added to the seed set. This simulates the user accepting the proposal. If a proposal does not exist in the gold standard it is added to the negative set. This is based on the idea of relevance feedback by B. Min and R. Grishman. [MG11] If the number of positive proposals is larger than the number of negative entities, the algorithm chooses random low ranked entities and adds them to the negative set, so that number of positive and negative proposals is equal. Therefore a new positive centroid is created based on the seeds and positive proposals and a negative centroid calculated from the negative proposals. This procedure is repeated 10 times, which results in 200 possible positive proposals.

[1] https://www.dea.gov/index.shtml
[2] https://www.nyu.edu/

The precision is the number of positive entities divided by the number of total proposals.

$$precision = \frac{\#pos\_entities}{\#pos\_entities + \#neg\_entities} = \frac{\#pos\_entities}{200} \tag{7.1}$$

7.1.2 Comparison

To get an impression of the performance of the newly implemented entity set expansion, four differently created entity sets vectors are considered.
The first entity set vectors are created by the original ICE system, which creates a huge set of sparse vectors, in which a similarity probability for every combination of words in the corpus is represented. Each vector is a sparse vector of the similarity of one word to any other word in the corpus. The vector is sparse as the similarity of one word to most other words is zero.
The second entity set vectors are the word-embeddings calculated by the implementation of this thesis. The vectors are calculated from the collection of nouns and context words appearing in the drug corpus by the DEA[3]. This is done via the 'Word2Vecf' algorithm. Therefore these vectors are the equivalent to the output of the original system.
The third entity set vectors are also acquired in the same way, but the context of words is calculated after removing all quantifiers from the corpus. By this, the hope is, that more drugs are recognized, because an entity existing of a quantifier plus a drug name won't appear in the gold standard, while the drug name on its own is available in the gold standard.
The forth entity set vectors are the pre-trained Google vectors obtained from the Google News dataset[al13]. These vectors are not trained on a specific field like the vectors trained on the drug data set, but are trained on news articles containing a broad spectrum of fields.
The intent of these four different entity set vectors is to show the advantages and disadvantages of word-embeddings and how the training on a specific domain affects the performance of the entity set expansion.

7.1.3 Evaluation Results Entity Set Expansion

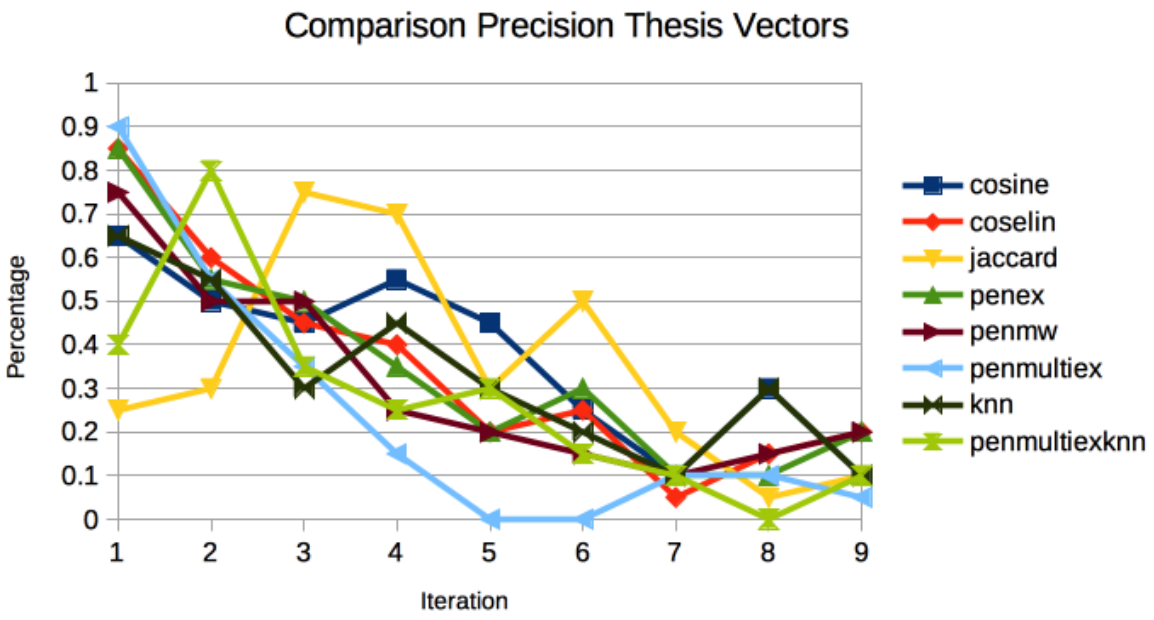
As described in Section 6.3, the entity set expansion methods are tested with different similarity functions. All entity set expansion methods are run on the drug entity set corpus and propose 20 entities in each of the 10 iterations.
This procedure is repeated for each of the different similarity functions. To ensure that

[3] https://www.dea.gov/index.shtml

kNN is using the optimal factor k, each entity set expansion method was run with k $\in$ (1, 9) for kNN as well as the combined multi-word & existing word penalty function applied with kNN.

The results of the different similarity functions are shown in table 10.1 for the original system, in table 10.7 for the system implemented in this thesis without the quantifiers removed, in table 10.13 for the system implemented in this thesis with the quantifiers removed and at last in table 10.19 for the existing Google vectors set.
As mentioned before in the comparison of the different similarity methods the optimal k for the kNN methods is used. The constant k is mentioned in the tables after the similarity function name. The evaluation of the constant k can be found in table 10.3 and table 10.5 for the original system, in table 10.9 and table 10.11 for the system implemented in this thesis without the quantifiers removed, in table 10.15 and table 10.17 for the system implemented in this thesis with the quantifiers removed and in table 10.21 and table 10.23 for the existing Google vectors set.
In each table the first column shows the iterations run by the system. The last row of an iteration shows the final score for the number of iterations while each previous row represents the precision of each run. The last row is the average value of all preceding results of an iteration.

Figure 7.1



An overall observation is, that all similarity functions have the highest precision in the first iteration and then in general decrease with each iteration except for Jaccard

similarity, which rises until it has its peak at three iterations and then decreases again as it can be seen in figure 7.1.
In figure 7.1, which shows the results for each iteration of the different similarity functions for the vectors retrieved by the implemented method without removing the quantifiers, the similarity functions cosine, existing word penalty function, multi-word penalty function, combined multi-word & existing word penalty function, and kNN are all following a similar pattern as they are all monotonously decreasing except for a maximum of two data points, which break the pattern.
While they all are decreasing with higher iterations, they still show noticeable differences. Note that each iteration can't propose entities, which were already proposed in earlier iterations. Therefore the first iterations are the ones with the highest precision for most similarity functions as they propose the entities, which are the most likely to be accepted by the gold standard.
For the first iteration the cosine similarity function with the combined multi-word & existing word penalty achieves the highest precision at 90%. This is due to the fact, that multi-word entities, which contain no existing word in the seed set are penalized heavily and multi-word entities, which contain words existing in the seed set, are penalized slightly. By that all single-word entities move up in the ranking. Therefore all single-word drugs will be ranked the highest and be pushed above the multi-word entities consisting of a drug name and an additional term. Because of the Distributional Hypothesis and the reduction of the risk to propose multi-word entities, which are more likely not to be a drug although their vectors are similar, the chance of proposing a wrong entity is significantly lower than in the other similarity functions. This explains the high recognition rate of drug names in the first iteration.
The problem with this approach is, that every single-word entity is pushed to the top in the ranking. As a consequence the recognition rate drops by 45% in the second iteration to 55% and after that even lower.
Also cosine similarity combined with the eliminating or penalizing the intersection between the entities in the corpus and the entities in the seeds is behaving in a similar manner as it can be seen in figure 7.1 by the lines 'coselin' and 'penex'. These similarities achieve 85% precision in the first iterations and then drop to 60% and 55% in the second iteration.
By that they achieve a similar average precision in the second iteration as the top ranked similarity function. After that they don't drop in their precision rates as low as the cosine similarity function with the combined multi-word & existing word penalty. An explanation for this behavior is that in the iterations starting from iteration three also multi-word entities become more important, because most of the single-word drug names are already recognized.
As the multi-word entities still get penalized by a large constant, multi-words which don't have any existing word in the gold standard will always be ranked low and won't be proposed to the user.

The next highest precision in the first iteration has cosine similarity with multi-word penalizing in general. As all multi-words are penalized by a large constant, also multi-word entities, which contain words from the seed set are ranked low. With about 65% precision and 50% and 55% cosine similarity and kNN with $k\bar{7}$ still achieve a reasonable result.
The kNN with combined multi-word & existing word penalty function on the other hand has a low recognition rate for the first iteration. This can be explained that all the multi-words are penalized. By that also the correct multi-words are ranked lower as the algorithm tries to recognize single word entities first, because most drugs in the gold standard are single word terms. If it then happens that not many correct drug entities are in close range around the seeds, it can only give proposals with low probabilities, which are likely to be wrong. That is, why the combined multi-word & existing word penalty function using cosine to calculate the similarity won't encounter that problem as it takes a look at all words, not only the ones in close range of the existing seed terms.
For Jaccard similarity the curve with a peak at three iterations makes sense. In the first iterations mostly multi-words including one of the two seeds are proposed as those will have a non-zero result of the union. All others have a similarity of zero. After a few iterations the algorithm hits entities available in the gold standard by luck. Following that, it can recognize combinations of the new found entities and other entities, which are recognized by the gold standard.
As this algorithm needs luck to find single-word entities and then finds the multi-word entities, which are containing these single-word entities and are in the gold standard, the algorithm produces poor results.

When having a look at high number of iterations all methods drop down to a low precision. The gold standard consists of 223 drugs, which could be found by the system. At nine iterations with 20 proposals each, the similarity functions achieve precision values between 37% and 24%. Cosine with 37% and cosine with eliminating intersection, cosine with penalizing multi-words and Jaccard with 35% all achieve similar results. Cosine with combined multi-word & existing word penalty function, which had the highest precision in the first iterations, returned the lowest average precision with 24%, when run for nine iterations.
While the results for all iterations are interesting, in an application for a real use case no user will be expected to look at 20 proposals for nine iterations by a system and decide on all of them if they are correct of not.
Also calculating recall does not make too much sense, as even a perfect algorithm could never reach 100% at three or even nine iterations with 20 proposals each, because the gold standard contains 223 entities, while the algorithm can only propose a combined number of 60 (180) entities in three (nine) iterations. Consequently the F1 score is not very meaningful either. Therefore recall and F1 score are mentioned in the interests of completeness int table 7.1, but won't be followed further.

Iterations	Recall	F1 Score
1	0.08	0.148
2	0.13	0.21
3	0.16	0.24
4	0.175	0.25
5	0.175	0.25
6	0.175	0.25
7	0.184	0.26
8	0.193	0.265
9	0.197	0.267

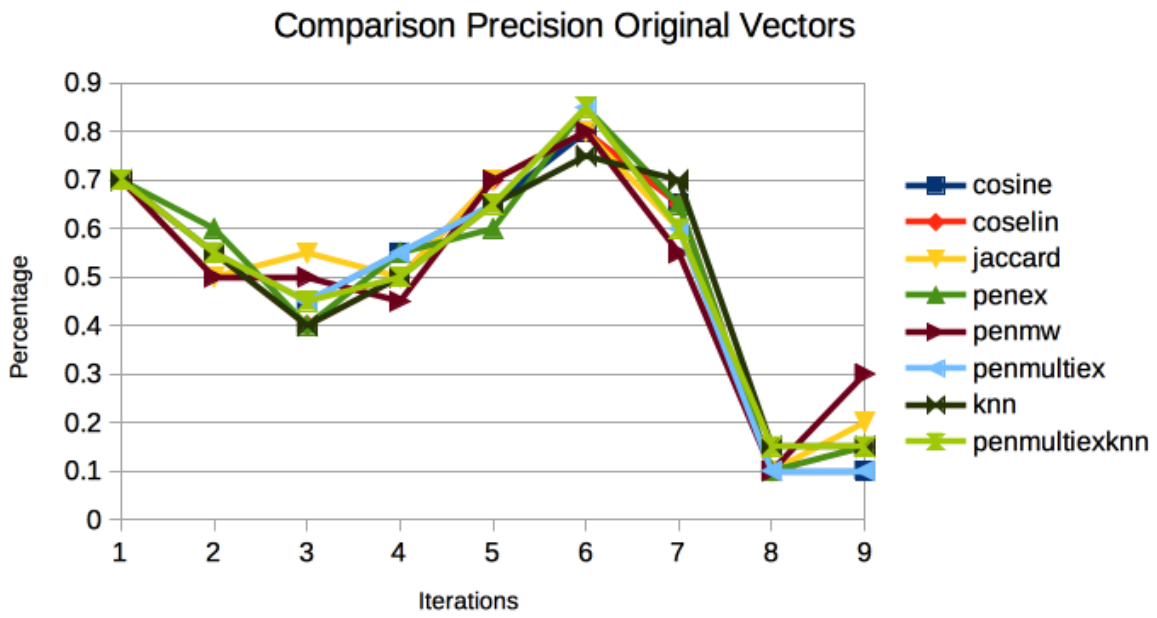
Table 7.1: Recall and F1 score for the cosine with combined multi-word & existing word penalty function

Training a system for three to four iterations is a more realistic choice. A user can be expected to look at 60 proposals and decide, whether they fit into the entity set or not. At three iterations the best similarity functions get a precision of 63%, at four iterations a precision of 58% is returned.

In comparison to these significant differences between the similarity functions used on word embeddings, all similarity functions behave similarly when applied to the sparse vectors of the original system. This is expected as in the original system each dimension is a evaluation of the similarity between the entity of the vector and another entity.
Therefore the ranking does not react too much to penalizing multi-word entities or existing-word entities as sparse vectors of this kind have a high value for similar words and for all others close to zero. That way even when penalizing an entity which has a similarity close to one, it will still be higher than the entities with values close to zero. Also kNN does not alter the results noticeable.
As shown in figure 7.2 all curves are close together and don't offer much difference.

The other two methods, using the embedded vectors without quantifiers and using the Google vectors, both produce similar results to method using the embedded methods with quantifiers. Without quantifiers the results are slightly better, achieving a precision of up to 90% in the first iteration on several similarity functions, up to 68% in the third iteration, up to 65% in the fourth iteration, and up to 47% in the ninth iteration.
Also the Google vectors, which weren't trained specifically on a drug related corpus, return good results. With these vectors the entity set expansion returns up to 85% correct entities in the first iteration, up to 70% in the third iteration, up to 63% in the fourth iteration, and up to 43% in the ninth iteration. This shows that it is not necessary to create vectors specifically trained only on a domain specific context to achieve good results.

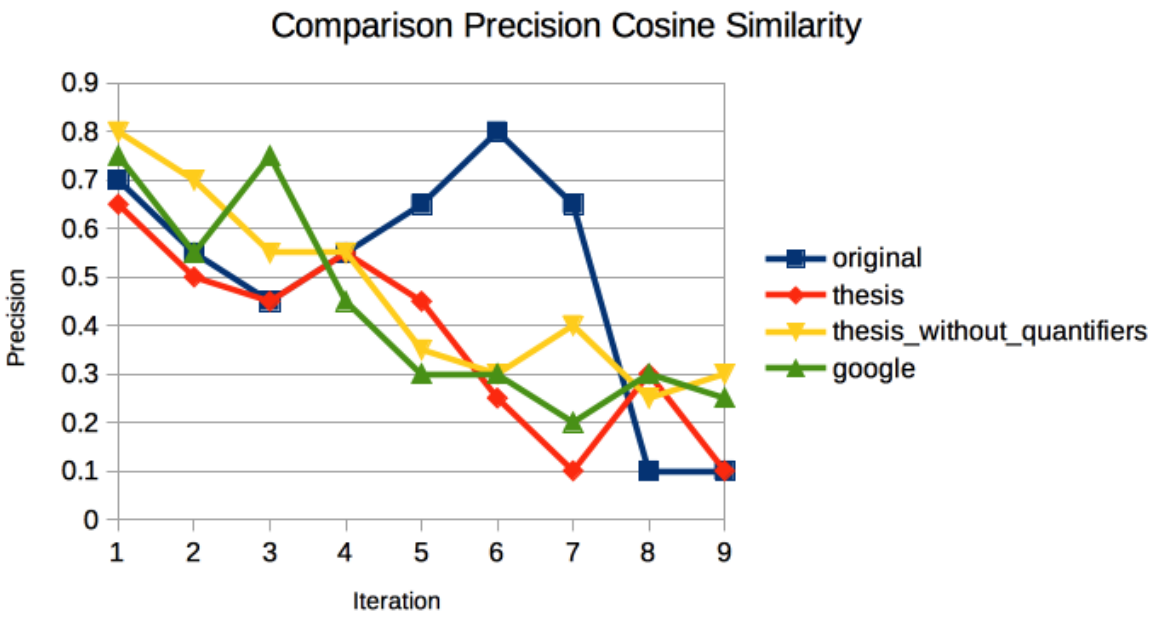
Figure 7.2



The Google vectors were trained on the Google News dataset, which has about 100 billion words, containing all sorts of information including news about drugs.[al13]

Therefore the Google vectors can be used on every domain existing in the Google News dataset. Only words which have different meanings in different contexts might be a problem for this corpus.
This is not the case when expanding entity sets on a drug corpus searching for drugs as it can be seen in figure 7.3, where the different precisions for each iteration for cosine similarity are displayed.
In each iteration the three methods using 'word2vec'-vectors achieve similar results with the Google vectors having an advantage of precision in iteration three, but then dropping under the precisions of the other two methods in the following iterations. To notice is, that the original, sparse vectors achieve a much higher precision in later iterations starting at iteration 5. An explanation for the higher precision is the usage of sparse vectors, in which the entity of the vector is compared in each dimension with one other entity. To find similar entities, it is possible to just sort all dimensions of the vectors in the seed set and propose those entities, which are at the top in the sorted dimensions across all seeds. Therefore it is more distinct, which entities are good predictions, as the system only has to check, which vectors are the highest in the dimensions of the entities of the seed set. I.e. if the drug 'cocaine' is compared with all other entities in the fifth

Figure 7.3

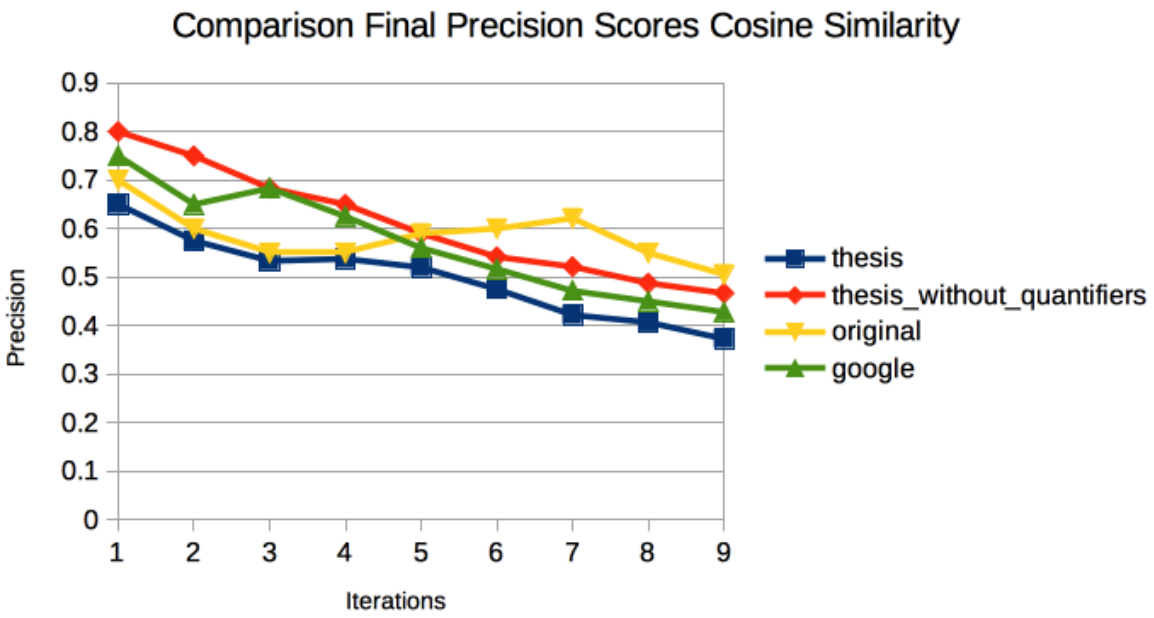


dimension of the vectors, the method can order all vectors in decreasing order according to the fifth dimension to find similar entities.

In all these figures each data point at iteration i is the precision of the i-th iteration. To see how the methods performed in total the average over the recent iteration with the results of all previous iterations has to be calculated. The average precision of the i-th iteration is shown in figure 7.4. Figure 7.4 shows, that for lower iterations the implemented method, which creates the word-embeddings without quantifiers, achieves the highest precision for the first five iterations and then the original, sparse vectors take the lead and achieve $\approx 51\%$ precision while the word-embeddings only offer $\approx 47\%$ precision in its proposals.

As all these results are based on cosine similarity, in figure 7.5 the final precisions are shown when multi-word entities and entities with existing words are penalized. As mentioned before, for the original, sparse vectors the similarity function only has a minor influence and the curve looks similar to the one of figure 7.4.
For the word-embedding methods the results are different. While they start at a higher precision (up to 90% for the implemented ones), it drops down to 22% in the ninth iteration. With these similarity functions the original, sparse vectors already achieve a higher precision in the fourth iteration. As it can be seen in this example, the expectations of a user for the system define, which vectors and similarity function is optimal for the given use case. If the user knows, that the corpus won't change in the future, then

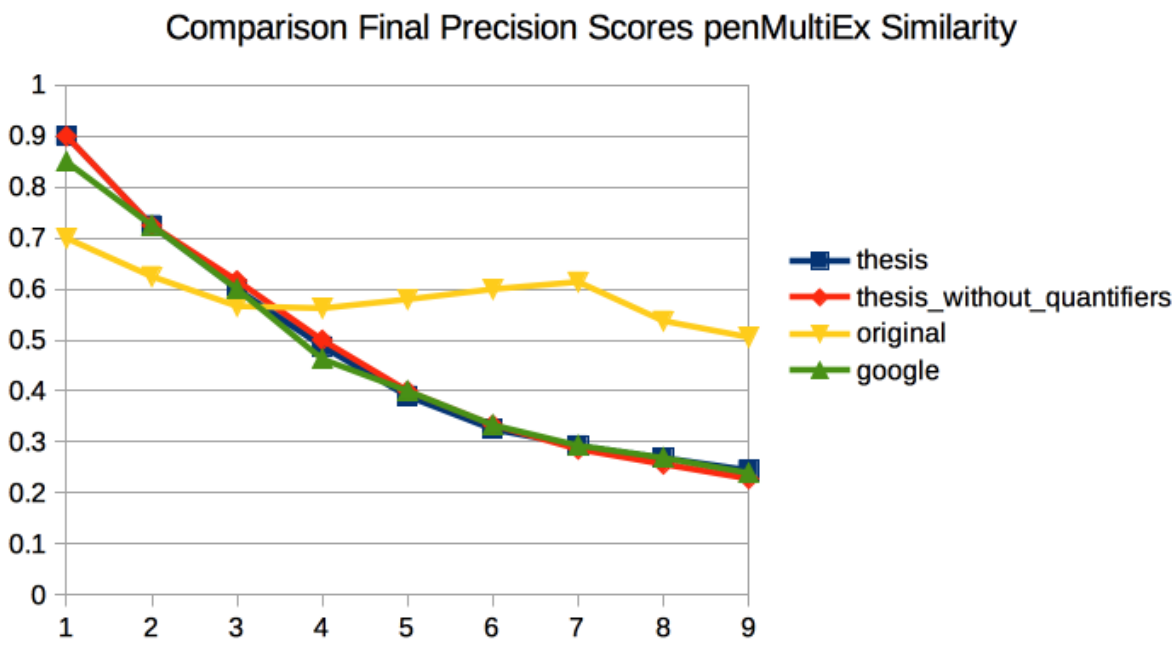
Figure 7.4



sparse vectors can be of advantage as they need less computational processing power. But as for cosine similarity the precisions are similar in the original, sparse vectors as well as the word-embeddings, the word-embeddings offer the advantage, that new documents can be added easily to the existing system. Also the computational expense has to be mentioned. While for the original system with the sparse vectors, it is a simple comparison to evaluate similar entities, for word-embeddings the similarity has to be calculated during runtime. With the presented similarity functions, this is not a huge problem, but the comparison of sparse vectors is faster.
If the user wants to save the time of preprocessing a corpus for entity set expansion, even an already existing vector set just like the Google vectors can be used.
If the user only wants to spend little time on training the model, a similarity function with high precision rates in the first iterations cosine similarity with penalization of multi-words and existing words can be used. Otherwise an unoptimized similarity function like the cosine similarity might offer more stability to the proposal results.
An unoptimized function like cosine similarity is also the better choice, if the use case is not specified.

In many companies the usage of a system will be very specific and then tuning the parameters of the chosen similarity is of advantage. This was showcases by tuning the similarity functions for drug entities, but can be achieved for any domain.

Figure 7.5



Another conclusion of this evaluation is, that the precision of the system is hard to improve after creating the word-embeddings. To increase precision, more advances have to be done before creating the word vectors, as it was done by combining multi-word entities by an underscore and by eliminating quantifiers from entities.

After proofing the usability and high precision of the proposed entity set algorithm, the next level of abstraction is the extraction of whole relations. Instead of dealing with single words, whole sentences are considered and the goal is to find similar relations within the corpus.

7.2 Evaluation Relation Extraction

After successfully improving and evaluating entity set expansion, also relation extraction is evaluated. For the relation extraction the same corpus about drug crimes is used as for entity set expansion. The gold standard is an already existing gold standard, consisting of input text and its correct relations for extracting relations of the type 'sell'. That means, that the system is expected to detect and extract all relations, which represent a sell-action.

In the first attempt, which was to look for appearances of the word 'sell' and extract all relations of the sentences, in which 'sell' appeared as the source or target word of a relation, only exact matches are recognized. This attempt does not look at the actual words, which are in relation with the word 'sell', but only looks at the POS tags. Even then it is highly unlikely, that sentences have the exact same POS tags in the exact same sentence structure appearing in a sentence. Therefore it is not unexpected, that the algorithm does not extract any other relations, when applied to the drug corpus. The only relations this algorithm finds, are the ones given as seed relations to the algorithm to start on. When running on the drug corpus, it does not find any similar sentence to the sentences, which contain the word 'sell'.
As expected this results in a precision and recall of 0% for the evaluation of the test corpus.
Another reason for the absence of any recognition is the relatively small corpus of drugs this thesis is working on and also the small test set of only 25 XML-files.
In a larger corpus it is more likely, that two sentences will occur in the exact same structure. But even then it will be a rarity and not lead to a high precision, which is the goal. The same applies also to recall. If it only detects exact matches the number of relations in the gold standard, which go undetected, will overwhelm.
To get more suggestion for a method is needed, which is less strict on the relations of the word and focuses more the actual surrounding words to extract relevant relations.

This is attempted in the second idea. Instead of looking at all the relations, in which the word 'sell' appears in a sentence, the idea is to look only at the subject and direct object of the word and extract these.
While the problem of the first attempt is, that is doesn't recognize any correct relations, the problem with this attempt is, that is accepts a lot of false positives, which isn't the case in the first attempt. In example 'sell' and 'buy' have opposing meanings, but are used in a very similar way. How word-embeddings methods are able to distinguish between such antonyms is shown in section 4.3.4. In general both verbs are used in combination with a person and a direct object. The algorithm has no way of telling a difference between the two actions as person and object can be the same entities for both verbs.
Therefore the algorithm would accept buy-relations as similar relations to sell.

To avoid these false positives, the word embeddings are used in a third attempt. The strategy is similar the the on of the second attempt. The only difference is, that before accepting verbs as new seeds, they are compared by their vectors using cosine-similarity. If the cosine-similarity returns a factor higher than the cutoff value k, then the new verb is added to the seed set, otherwise it is discarded. By comparing the new verb with the centroid of the verbs in the seed set, it is possible to recognize similar words according to the Distributional Hypothesis. 'sell' and 'buy' still have a higher similarity than 'sell' and 'swim', but synonyms of 'sell' are ranked higher than 'buy', because they have more

context words in common.
The factor k is critical for the performance of the method. This leads to an optimization problem of k. The higher factor k is, the higher is the precision, but the lower is the recall as it can be seen in the tables 7.3, 7.4, 7.5.
In all the three tables presented, the factor k is evaluated for $k \in [0.1; 1]$, but the model is trained on different sized training sets. As shown in the three tables, the results for recall and precision are closely related to the cutoff value k, even though k is not the only significant influence on them. The higher k is, the less verbs are accepted as similar to 'sell' and as a consequence less wrong relations are proposed. This leads to a higher precision, but a lower recall, because if there are less verbs in the seed set, the number of different relations, which are accepted by the algorithm, is lower. With a lower number of relations accepted, the chance to find the relations in the test set is lower as well.
In table 7.2 the verbs extracted for the test set at a cutoff of 0.5 are shown. Due to the lack of lemmatization before extracting relations, verbs appear more than once in different versions. This has influence on recall and precision. In this table the problem

Verbs in the centroid
sell
sold
acquire
distribute
purchased
distributed
shipped
acquired
supplied
distributing
manufactured
supplying

Table 7.2: Verbs in the centroid for the test set at a cutoff of k=0.5

of antonyms can be seen. While all verbs are definitely related to the word 'sell',acquire' and 'purchase' are more similar to 'buy' and therefore are an antonym to 'sell'. As recall and precision are at their maximum when the other is at its minimum, a cutoff value in between has to be chosen.
The F1 score measures the accuracy of the system and is the harmonic mean of precision and recall. Therefor it is a good indicator for an optimal system, because the goal of any system is to achieve both high results for precision and recall. The F1 score reaches its maximum in all three cases close to 0.7. For 7.3 and 7.4 the maximum F1 score is at k=0.693. For 7.5 the maximum F1 score is at 0.7. Also the maximum F1 score does not

differ much in all three table, but is at about 0.1.
In all cases the maximum F1 score is reached at a recall of 6.25% and a precision of about 21 to 27%.
While precision can be pushed to 100% at a cutoff value of 0.75 or higher, recall is the highest at 7.8% at a cutoff value of 0.5 or lower. This shows, that comparing the word-embeddings of words is a good way to improve precision, but the way, that the algorithm tries to match relations has to be changed in order to improve recall and achieve a useful F1 score.
Even for low cutoff values recall does not increase over 7.8%, but other papers show, that much higher recall values are possible. For example in [SGD16] recall results of 60 to 90% are achieved. Therefore it would be very interesting to combine the idea of comparing the verbs of the new relations with the already existing verbs in the seed set and the method of an algorithm, which achieves good recall results to see if that improves the precision even more.

Test Set 100			
k	Recall	Precision	F1 Score
1.0	0.03125	1.0	0.06060606060606061
0.9	0.03125	1.0	0.06060606060606061
0.8	0.03125	1.0	0.06060606060606061
0.7	0.03125	1.0	0.06060606060606061
0.694	0.03125	1.0	0.06060606060606061
0.693	0.0625	0.26666666666666666	0.10126582278481013
0.6	0.0625	0.26666666666666666	0.10126582278481013
0.58	0.0625	0.26666666666666666	0.10126582278481013
0.57	0.0625	0.25	0.1
0.56	0.0625	0.23529411764705882	0.09876543209876543
0.5	0.078125	0.09090909090909091	0.08403361344537816
0.4	0.078125	0.017241379310344827	0.02824858757062147
0.3	0.078125	0.007215007215007215	0.01321003963011889
0.2	0.078125	0.0044603033006244425	0.008438818565400843
0.1	0.078125	0.004351610095735422	0.008244023083264633

Table 7.3: Evaluating Recall, Precision, and F1 score, when training on train set with 100 Files, for different k

In general the similarity measurement offers some advantage in comparison to the first and second approach. As this method is similar to the second attempt, it is also more flexible and recognizes more matches than the first attempt. It offers the user the opportunity to choose, if precision or recall is more important by setting k accordingly,

Test Set 1000			
k	Recall	Precision	F1 Score
1.0	0.03125	1.0	0.06060606060606061
0.9	0.03125	1.0	0.06060606060606061
0.8	0.03125	1.0	0.06060606060606061
0.7	0.03125	1.0	0.06060606060606061
0.694	0.03125	1.0	0.06060606060606061
0.693	0.0625	0.2222222222222222	0.0975609756097561
0.692	0.0625	0.21052631578947367	0.0963855421686747
0.6	0.0625	0.21052631578947367	0.0963855421686747
0.56	0.0625	0.21052631578947367	0.0963855421686747
0.55	0.0625	0.19047619047619047	0.09411764705882353
0.54	0.0625	0.19047619047619047	0.09411764705882353
0.53	0.0625	0.19047619047619047	0.09411764705882353
0.52	0.0625	0.0851063829787234	0.07207207207207207
0.5	0.078125	0.08196721311475409	0.08
0.4	0.078125	0.024509803921568627	0.03731343283582089
0.3	0.078125	0.004901960784313725	0.00922509225092251
0.2	0.078125	0.003518648838845883	0.006734006734006734
0.1	0.078125	0.003259452411994785	0.00625782227784731

Table 7.4: Evaluating Recall, Precision, and F1 score, when training on train set with 1000 Files, for different k

and by that setting how many false positives are acceptable. As shown above the harmonic mean of precision and recall is maximized at a cutoff value of about 0.7.

A problem, that all attempts have, is that the corpus is not lemmatized, before extracting relations. This leads to the fact, that 'sell' and 'sold' are two different entries in the seed set. While this, is no problem, other than unnecessary memory usage, when looking at the seed relations, skipping lemmatizing can be a serious problem. For example the nouns 'Dealer' and 'dealer' are different entries in the seed relations as well as 'dealer' and 'dealers'. This means, that if only 'dealer' exists in the seed relations, the algorithm won't recognize 'dealers', Dealer', or 'Dealers' in the test corpus. This could improve both precision and recall significantly.

TestSet Complete			
k	Recall	Precision	F1 Score
1.0	0.03125	1.0	0.06060606060606061
0.9	0.03125	1.0	0.06060606060606061
0.8	0.03125	1.0	0.06060606060606061
0.75	0.03125	1.0	0.06060606060606061
0.74	0.03125	0.6666666666666666	0.05970149253731343
0.71	0.03125	0.6666666666666666	0.05970149253731343
0.7	0.0625	0.21052631578947367	0.0963855421686747
0.6	0.0625	0.21052631578947367	0.0963855421686747
0.54	0.0625	0.21052631578947367	0.0963855421686747
0.53	0.0625	0.2	0.09523809523809523
0.52	0.0625	0.08695652173913043	0.07272727272727272
0.51	0.078125	0.07462686567164178	0.07633587786259542
0.5	0.078125	0.07462686567164178	0.07633587786259542
0.4	0.078125	0.02577319587628866	0.03875968992248062

Table 7.5: Evaluating Recall, Precision, and F1 score, when training on complete train set with 6391 Files, for different k

8 Workflow

8.1 Introduction ICE

In this chapter the workflow of the IES will be explained. Therefore the customizable IES ICE will be described and changes made to the system will be highlighted.
It originally was designed to extend the information extraction engine Jet.
Jet is a toolkit, that is written for the English language. Its core idea is to work with annotations for the features it offers. Each annotation consists of a text span of a document, the type of annotation and a set of attributes of the annotation. Jet is used for language analysis supporting features including tokenizing, sentence splitting, part-of-speech tagging, parsing, dependency parsing among others. Its features can be added to a pipeline and are then executed for the given document. ICE offers two approaches to interact with the system to execute preprocessing. It has a GUI and a command line interface (CLI). Both are equally powerful.

8.2 Overview Workflow

This system aims at entity set expansion and relation extraction from unstructured text. To achieve this, the system has to preprocess the unstructured text first. The user can access the system via two ways for preprocessing.
He can either use the CLI or the GUI. For entity indexing and entity set expansion as well as relation extraction, the user has to use the GUI. The GUI gives the option of entering information at runtime, such as a cut-off limit for the indexing step, names for entity sets, seeds in the entity expansion step and relation names and seeds in the relation extraction step. For the preprocessing step, the user has to enter the directory path for the files, which shall be preprocessed, he has to specify the input format and has to choose a background corpus, if a corpus already exists.
In the preprocessing step the corpus is analyzed by using tokenization, sentence splitting, part-of-speech tagging, named-entity recognition, and parsing/dependency parsing.
While all following steps - after the preprocessing - are language independent, the preprocessing has to use language dependent libraries.

Therefore the system has to be extended to choose the language and libraries needed for the languages at runtime. For each language a language specific parser is needed, as each language is based on different grammatical rules and these rules are needed to extract dependency structures from sentences, and a dictionary like the Penn Treebank is needed to tag each entity with its according POS tag.
To accomplish the choice of different languages during runtime and still make entity set expansion and relation extraction compatible with all of them, the outputs of the preprocessing steps have to be standardized.
To open the system also to future language packages, an interface is implemented, which sets the required methods and their outputs. By that, the standardized outputs of the preprocessing phase is enforced. To choose the language during runtime Java Reflections is used. With Java Reflections Java classes can be set during runtime. The interface is also is essential for Java Reflections, to enforce that the needed functions in the preprocessing classes extending the preprocessing interface are available.
The results of any of the preprocessing classes are saved to six different files.

One file for each of the following properties:

1. names/proper nouns
2. noun phrases
3. part-of-speech tags
4. dependencies
5. automatic content extraction
6. entity mentions

The files, which are relevant for this master thesis, are the POS tags and the dependency annotations, which are the syntactic relations. In the preprocessing step the corpus name, the number of documents in the corpus, the directory of the input files, the filter, the preprocessor used for preprocessing, and the background corpus are saved to 'ice.yml'. This has the advantage, that for each corpus all necessary annotations and properties can be loaded without having to preprocess all input data again.
The preprocessing step is a time consuming process and therefore re-computations should be avoided. Saving the used preprocessor is also necessary to be able to differentiate between the results of different libraries for English, and also to use the same libraries in later tasks. I.e. in the current system in the evaluation of relation extraction the test file has to be annotated again to extract relations using the trained model and compare it to the gold standard.

After the preprocessing step is finished, the entities of the corpus have to be indexed. This is needed in order to compare entities to each other. The user has to set a cutoff

value, which determines which entities should be considered.
Entities, which occur less often than the cutoff, won't factor in. The entity set indexer calculates a vector for each noun in relation to all entities. Here either the standard indexer is used, which builds a sparse vector by estimating a similarity value for each entity in the corpus compared to any other entity in the corpus or the 'Word2Vec' model is used, which calculates a dense vector of a set dimension using the contexts extracted in preprocessing.
While the first option doesn't need further computations to get the similarity between two entities and simply can order all entities according to the i-th dimension in the sparse vector, in which the source entity is compared to target entity. For the word-embeddings given by the 'Word2Vec' model the similarities have to be calculated using a similarity function during runtime.
This causes a higher computational complexity during runtime than for the sparse vectors of the original indexer. The resulting vectors of both models are saved to a file, so that they can be loaded again without reprocessing the indexing. The system also saves the used indexing class and the correlated expanding class to 'ice.yml'. This is important for the expander as the output of the indexers is not standardized. Standardizing the output is not an option as different methods are using different formats of storing the indexing results.
The 'Word2Vec' method needs a dense vector as the vector dimensions are not directly correlated to an entity, but represent features learned by the machine learning model during training time. That way most values in the vectors will be non-zero values.
On the other hand in the original indexer, each dimension is a comparison between the entity of the vector and another entity of the vocabulary. That way, there will be many values at zero or close to zero. In this case using a dense vector is a waste of resources and a sparse vector is a much better option.
But the output could also be stored in a table or any other data structure. As long as a compatible scoring function is implemented, there are no restrictions to how the entities are ranked.

With the entities indexed, the system is ready to perform entity set expansion. To create an entity set, the user can choose the name and seeds itself or let the system suggest similar seeds and name an entity set for them. The user can always add seeds himself, but he also has the option to use the 'suggest'-functionality of the system to offer entities, which might match the entity set.
To suggest entities, the system ranks all entities according to the similarity function, which is set in 'ice.yml'. Any similarity function can be used as long as it is compatible with the output format of the chosen indexer. I.e. cosine similarity takes two vectors as input to multiply them using the dot product and normalizing them with the multiplications of the length. The ranking method returns the entities in a ordered list starting with the most similar words. From the list suggested by the system the user chooses

which entities are correct, wrong or neutral and therefore set the tags ’Yes’, "No’, and ’Undecided’ for each word. Those tags will be used for ranking and re-ranking the order of the suggested words in the list to show the user the words, which match the entity set with the highest probability.

After building up entity sets, phrases and sentential phrases can be calculated. Those will be used to extract relations. The user has the option to either add new relations himself or let the system suggest a relation for him. Just like in entity set expansion the system ranks all relations extracted from the corpus according to a similarity function. For the relations the system offers the capability of suggesting similar relations. The user interface for relation extraction is similar to the entity set expansion. The user has the choice of entering a relation of his choice or use the suggest functionality of the system. The system is able to suggest new relations with the described method in 6.5.
To show the differences made to the original ICE, a comparison of the preprocessing, entity set expansion and relation extraction is given. Also differences between the German and English pipeline provided by Stanford CoreNLP is provided.

8.3 Comparison original and new system

8.3.1 Preprocessing - Comparison of JET and Stanford CoreNLP Toolkit libraries

For the preprocessing, originally Jet was used. In order to offer more flexibility, the new system uses Java Reflections to choose the preprocessing tool at runtime. Java Reflection is a feature that allows to “examine and manipulate a Java class from within itself.”[1]
An interface was created to enforce all necessary methods are set for the preprocessing step and to standardize the output of the preprocessing step. The implemented system showcases working preprocessing classes using the original Jet and a newly implemented preprocessor based on Stanford CoreNLP.
The Jet pipeline uses tokenization, sentence splitting, lexicon lookup, time expression recognition and name tagging using Hidden Markov Model.[Gri]
The Stanford CoreNLP pipeline for English or German uses tokenization, sentence splitting, POS tagging, named-entity recognition (NER), dependency parsing, and syntactic analysis.
For English texts also co-referencing, entity mentions, gender tagging, and sentiment analysis are available. The functions of the pipelines features are:

[1] http://www.oracle.com/technetwork/articles/java/javareflection-1536171.html

- Tokenize is the method to recognize tokens within the given text.
- Sentence splitting splits a sequence of tokens extracted from the text into sentences.
- Lexicon lookup assigns a lexical entry to each POS tag in the annotated text.
- Lemma lemmatizes all words. By lemmatizing the size of the dictionary is reduced and matching entities is easier. The construction of the syntactic dependency tree is still possible in the same way.
- NER annotates real world entities from the unstructured text. (named entities: PERSON, LOCATION, ORGANISATION, MISC; numerical entities: MONEY, NUMBER, DATE, TIME, DURATION, SET)
- Parse parses the text based on a probabilistic syntactic analysis.
- Constituency Parsing splits a text into sub-phrases to build a tree of the text.
- Dependency Parsing builds up a tree containing the relationship structure of the input text.
- Sentiment Analysis, also known as opinion mining, extracts the subjective meaning of a text.
- Mention Detection detects when a mention is used.
- Co-reference detects, which entity is meant by a used mention. This might be a pronoun, but can also be a synonym among others.
- Open Information Extraction (Open IE) extracts relation tuples from plain text.

[MSB+14]

The current setup uses mention tagging by the Jet pipeline also for the Stanford CoreNLP pipeline implementation. All three preprocessing pipelines are used to get entity names, POS tags, noun phrases, and relations. While there is no imminent advantage of using Stanford CoreNLP over Jet for English, Stanford CoreNLP offers libraries for a variety of languages: English, Arabic, Chinese, French, German, and Spanish. Therefore it is of advantage for the purpose of this work. As said before, it is not necessary to take libraries of the same source, but any preprocessing tool can be used in combination with this system. I.e. it is also possible to take Jet as tool for English preprocessing and Stanford CoreNLP to preprocess German. As long as the toolkit used provides all features used by the interface and is able to create the correct outputs, it can be used.

8.3.2 Comparison pipeline features of different languages in Stanford CoreNLP toolkit

Not all languages support all features of the pipeline yet. For example the implementation of this work uses NER. Therefore the system can only be used with Chinese, English, German and Spanish currently. As table 8.1 shows, Arabic and French don't support NER. It has to be accomplished by another library or implemented by the user.

Annotator	AR	ZH	EN	FR	DE	ES
Tokenize	y	y	y	y	y	y
Sentence Split	y	y	y	y	y	y
Part of Speech	y	y	y	y	y	y
Lemma			y			
NER		y	y		y	y
Parse	y	y	y	y	y	y
Constituency Parsing	y	y	y	y	y	y
Dependency Parsing		y	y	y	y	
Sentiment Analysis			y			
Mention Detection		y	y			
Co-reference		y	y			
Open IE			y			

Table 8.1: Availability of annotators for different languages [MSB+14]

8.3.3 Differences of the indexers

Also for the indexing an interface is used in order to offer the needed flexibility for using different indexers.
The original indexer builds up sparse vectors with the dimension of the number of different entities in the corpus. In each dimension is the similarity value of one word of the vector and another word in the corpus displayed.
On the other hand the new implemented indexer creates word-embeddings, which are used to measure similarity between entities. These word-embeddings are stored as dense vector of a chosen dimension. (In this thesis the dimension was set to 300.) The dimensions are not directly linked to the similarity of the vectors entity and a specific entity, but are retrieved by the machine learning models 'word2vec' or 'word2vecf'.[al13] The word-embeddings are linked to features instead of words.

'word2vecf' offers the advantage, that it can run multiple iterations.[Gol14] With multiple iterations the learning process is more precise, but has the risk of over-fitting. Over-fitting a model means, that the function which was learned by the model, only has a very small threshold of accepting deviation from the data points in the learned model. For similarity measurements that means, that it will only accept very few other entities other than the entity itself, as only exact synonyms have the exact same context.
Performance-wise the original system is faster in retrieving the similarity between entities as it only has to read the dimension, in which the entity, for which similar entities are searched, is saved. That means, that the original system only has to read one position of the vector of each entity once and can rank all entities right away.
In 'word2vec' and 'word2vecf' similarity has to be calculated for every comparison when needed. Precision-wise the embedded methods provide an advantage for semi-supervised learnings at the first few iterations, while the original system has higher precision at higher iterations as shown in subsection 7.1.3 of section 6.2.

8.3.4 Differences of relation extractors

While both the original as well as the newly implemented method follow their Snowball method of Agichtein and Gravano, the original method uses a semi-supervised approach to extend the relation set, while the method of this thesis proposes an unsupervised approach. This reduces the dependency of human interaction and therefore avoids a slowdown or downtime enforced by waiting for user input. On the other hand, it increases the probability of adding wrong relation sets to the seed set. Also the stop criteria is different as a consequence of the supervision. In the semi-supervised approach the number of iterations for the user is critical for the accuracy of the system. In the unsupervised approach the systems checks, if there are new relations proposed. In this thesis the system stops, if it did not find any new relations for 3 iterations. Apart from the approach, the original method uses the dependency tree calculated by Jet and the lemmatized version of the words to detect new relations. On the other hand the method, proposed in this thesis, uses the dependency tree and the word embeddings to check, how probable it is, that a proposal actually is a synonym to one of the entries in the seed set. This offers the opportunity to be less strict in accepting dependency structures as the wrong relations are rejected by the similarity measurement of the word embeddings. The similarity measurement has a huge impact on the runtime of the system. The less strict the system is, the more relations have to be checked on their similarity. Also the size of the training corpus influence the runtime in an exponential way. The larger the training corpus, the more relations will be found, this leads to a larger seed set and then it has to recheck the whole corpus again for new possible candidates. In the end, it depends on the use case, which approach is the one to choose. If the user always wants to have the same relations, then the semi-supervised approach is a good solution

to achieve a high precision and recall. On the other hand, if the system will be used for various purposes and the relations are varying, it will be uncomfortable to need human input for every new relation in order to train the system on that purpose. In that case, a unsupervised approach is suited better.

With entity set expansion and relation extraction presented and evaluated and a multilingual approach successfully implemented, the research of this thesis is accomplished.

9 Conclusion

The research of this work focused on three aspects. The first one was how to improve entity set expansion by using the word-embedding method presented by Mikolov et al. and mainly focus on the optimization of similarity functions. The second one was how to improve relation extraction and also used Mikolov's method to improve precision. The third one was to develop a system, which can handle multiple languages while using the same methods with different language specific libraries. To achieve progress on entity set expansion and relation extraction, first the corpus had to be preprocessed to get the entities and the dependency trees. A new entity set expansion method for ICE was presented in combination with seven similarity functions, of which four are new custom functions optimized on the domain used in this thesis.
It was shown, that the optimized similarity functions, which were domain specific, offer the highest precision in the first few iterations, but after that fall back behind the standard similarity function. The cosine similarity with combined multi-word & existing word penalties achieved a recognition rate of 90% in the first iteration, when applied to the domain specific word-embeddings.
It was also shown, that the similarity functions work well on any word-embeddings, but have no influence on the sparse vectors, which represent a similarity of between two entities in each dimension.
Another insight retrieved in this work, was that domain specific word-embeddings work the best in the first iterations, while the sparse vectors dominate when conducting more or equal than 5 iterations.
On the other hand, also domain unspecific vectors like the Google vectors, which are available online for free, offer a high precision as they are trained on a huge news data set including many domains.
In the end a combination of preprocessing the extracted entities of the corpus, by removing all quantifiers and combining multi-word entities with an underscore, and an optimized similarity function for a specific domain produced the highest precision and shows, that it is worth to customize a system on the specific needs of the user instead of using a general use system. Also the proposed similarity function, in which adjectives and other words, which don't change the meaning of an entity, are removed from the entity

In the research on relation extraction the word-embeddings showed great potential to improve precision. Even though the relation extraction itself did not proof to be well suited, the word-embeddings enforced a high precision without damaging the recall too much. Precision and recall were reduced by not lemmatizing before extracting relations and by only accepting a simple grammatical structure of a verb in relation to a subject and a direct object. A further approach would be to test the idea of improving precision by comparing word-embeddings in combination with the path-based and distributional approach presented in [SGD16] by Shwartz et al., who achieved a recall of 0.89. As they already presented a high precision at the recall value of 0.89, precision could be pushed to 100% without damaging the recall too much.

And as last it was shown in this thesis, that a multilingual approach is achievable by using interfaces and setting the language plus its needed libraries and functionalities during runtime by Java reflections[1]. As the word-embeddings are not created in a language dependent method by using 'Word2Vecf', but only use the context given to it, all language specific methods have to produce a standardized output and from there on all methods word language independent. In future work, the multilingual approach could be extended to more languages and also to detect the language of the text automatically - in example by analyzing a random sentence of each input text with the language specific preprocessing methods and comparing the produced entities with the vocabularies of the correlated languages. Comparison, which achieves the highest precision, is the language of choice.

Finally, this thesis presented a combined system, in which multiple languages can be preprocessed, then entity sets can be expanded with a high precision and relations similar to a given seed relation can be extracted.

[1] http://www.oracle.com/technetwork/articles/java/javareflection-1536171.html

10 Appendix

Figure 10.1

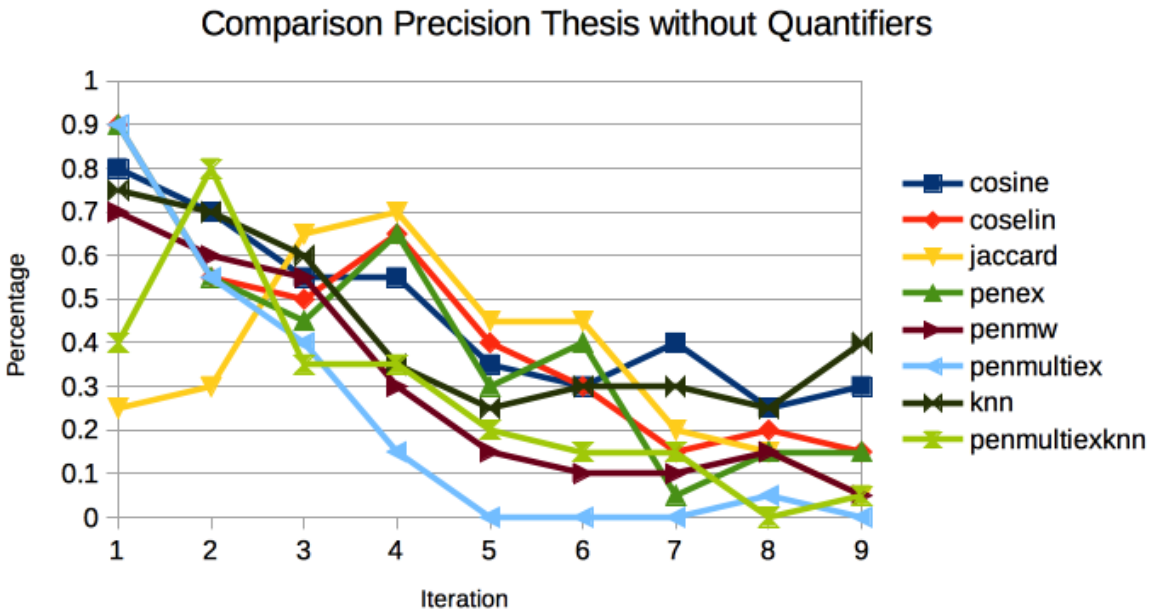


Figure 10.2

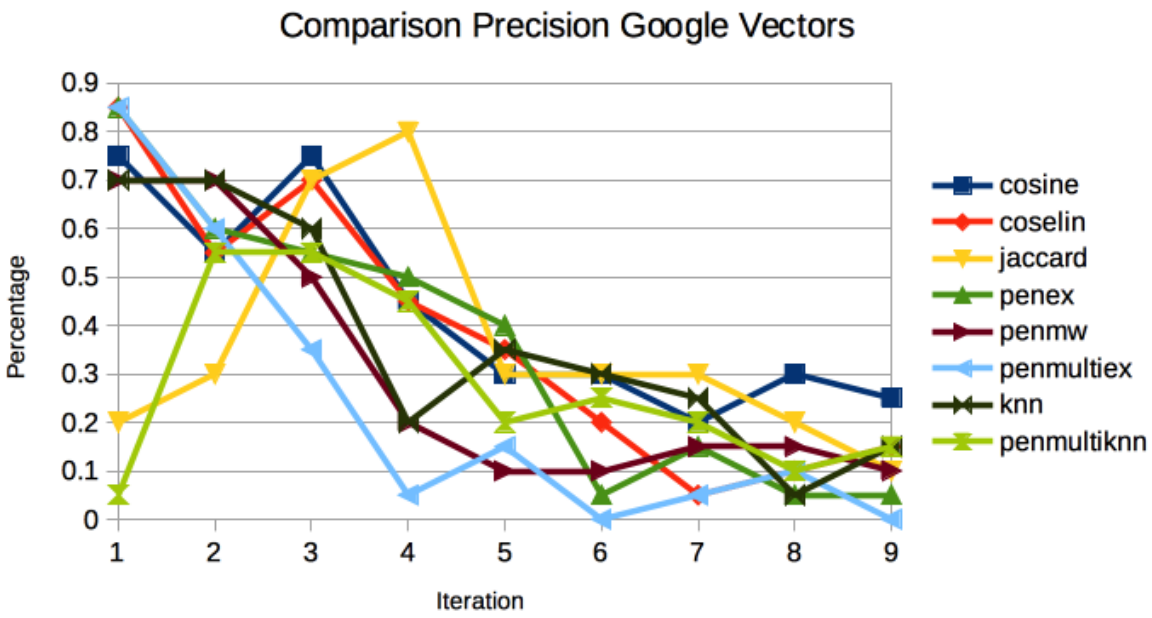

Comparison Precision Google Vectors
Percentage
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6
7
8
9
Iteration
cosine
coselin
jaccard
penex
penmw
penmultiex
knn
penmultiknn

Table 10.1: Comparison original vectors

Iterations	cosine	cosElIn	jaccard	penEx	penMW	penMultiEx	knn - 7	penMultiExKnn - 5
1	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
2	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.5	0.55	0.55	0.55	0.55	0.55	0.5	0.55
	0.6	0.63	0.63	0.63	0.63	0.63	0.6	0.63
3	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.55	0.55	0.6	0.55	0.55	0.55	0.55
	0.4	0.45	0.45	0.4	0.45	0.45	0.45	0.4
	0.55	0.57	0.57	0.57	0.57	0.57	0.57	0.55
4	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55
	0.4	0.4	0.45	0.4	0.4	0.45	0.45	0.35
	0.55	0.5	0.55	0.55	0.5	0.55	0.6	0.55
	0.55	0.54	0.56	0.55	0.54	0.56	0.58	0.54
5	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.5	0.55	0.5	0.6	0.5	0.55
	0.45	0.4	0.5	0.35	0.55	0.4	0.5	0.4
	0.5	0.5	0.55	0.55	0.45	0.6	0.55	0.5
	0.75	0.6	0.7	0.6	0.6	0.6	0.7	0.6
	0.59	0.56	0.59	0.55	0.56	0.58	0.59	0.55
6	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.55	0.5	0.55	0.55	0.55	0.55	0.5
	0.45	0.45	0.45	0.4	0.45	0.45	0.4	0.5
	0.55	0.5	0.55	0.5	0.55	0.55	0.5	0.5

	0.6	0.65	0.75	0.7	0.6	0.5	0.7	0.75
	0.75	0.8	0.8	0.8	0.8	0.85	0.8	0.8
	0.6	0.61	0.63	0.61	0.61	0.6	0.61	0.63
7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.5	0.55	0.55	0.6	0.6	0.55	0.55	0.55
	0.5	0.4	0.45	0.4	0.35	0.45	0.35	0.4
	0.6	0.45	0.55	0.5	0.55	0.55	0.6	0.55
	0.65	0.65	0.55	0.5	0.55	0.65	0.65	0.6
	0.75	0.8	0.85	0.7	0.8	0.85	0.8	0.8
	0.65	0.65	0.65	0.85	0.75	0.55	0.65	0.65
	0.62	0.6	0.61	0.61	0.61	0.61	0.61	0.61
8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.5	0.55	0.55	0.55	0.55	0.55	0.5	0.45
	0.45	0.45	0.45	0.4	0.45	0.45	0.35	0.6
	0.45	0.55	0.55	0.5	0.55	0.45	0.55	0.55
	0.7	0.7	0.65	0.7	0.65	0.7	0.6	0.6
	0.75	0.8	0.75	0.8	0.8	0.8	0.75	0.85
	0.7	0.6	0.7	0.65	0.65	0.6	0.8	0.65
	0.15	0.1	0.1	0.1	0.1	$5 \cdot 10^{-2}$	0.1	0.1
	0.55	0.56	0.56	0.55	0.56	0.54	0.54	0.56
9	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.55	0.5	0.6	0.5	0.55	0.55	0.55
	0.45	0.4	0.55	0.4	0.5	0.45	0.4	0.45
	0.55	0.5	0.5	0.55	0.45	0.55	0.5	0.5
	0.65	0.7	0.7	0.6	0.7	0.65	0.65	0.65
	0.8	0.8	0.8	0.85	0.8	0.85	0.75	0.85
	0.65	0.65	0.6	0.65	0.55	0.6	0.7	0.6
	0.1	0.1	0.1	0.1	0.1	0.1	0.15	0.15

	0.1	0.15	0.2	0.15	0.3	0.1	0.15	0.15
	0.51	0.51	0.52	0.51	0.51	0.51	0.51	0.51
10	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.55	0.5	0.55	0.55	0.5	0.6
	0.4	0.4	0.45	0.55	0.45	0.4	0.5	0.4
	0.5	0.55	0.6	0.45	0.5	0.5	0.5	0.55
	0.6	0.5	0.6	0.7	0.65	0.65	0.7	0.55
	0.8	0.85	0.8	0.8	0.8	0.8	0.85	0.75
	0.65	0.75	0.65	0.6	0.65	0.65	0.5	0.7
	0.15	0.1	0.1	0.2	0.15	0.15	0.15	0.2
	0.25	0.1	0.15	$5 \cdot 10^{-2}$	0.15	0.15	0.2	0.2
	0.25	0.15	0.25	0.2	0.1	0.25	0.3	0.2
	0.49	0.47	0.49	0.48	0.47	0.48	0.49	0.49

Table 10.3: Comparison original vectors on different KNN

Iteration	1	2	3	4	5	6	7	8	9
1	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
2	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.55	0.5	0.5	0.55	0.55	0.5	0.6	0.55
	0.63	0.63	0.6	0.6	0.63	0.63	0.6	0.65	0.63
3	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.6	0.55
	0.45	0.45	0.4	0.35	0.4	0.45	0.45	0.4	0.45
	0.57	0.57	0.55	0.53	0.55	0.57	0.57	0.57	0.57
4	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.55	0.6	0.6	0.55	0.55	0.55	0.6
	0.45	0.4	0.45	0.4	0.4	0.4	0.45	0.45	0.35
	0.6	0.55	0.5	0.55	0.55	0.55	0.6	0.55	0.5
	0.58	0.56	0.55	0.56	0.56	0.55	0.58	0.56	0.54
5	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.5	0.55	0.55	0.55	0.55	0.6	0.5	0.55	0.55
	0.6	0.45	0.45	0.35	0.4	0.4	0.5	0.4	0.45
	0.5	0.6	0.55	0.55	0.5	0.45	0.55	0.5	0.6
	0.7	0.55	0.65	0.65	0.7	0.35	0.7	0.65	0.6
	0.6	0.57	0.58	0.56	0.57	0.5	0.59	0.56	0.58
6	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.6	0.55	0.55	0.6	0.55	0.5	0.55	0.55	0.6
	0.4	0.45	0.4	0.35	0.45	0.55	0.4	0.45	0.4
	0.5	0.6	0.5	0.5	0.6	0.45	0.5	0.55	0.5

	0.55	0.55	0.65	0.65	0.5	0.6	0.7	0.65	0.35
	0.8	0.85	0.8	0.8	0.85	0.8	0.8	0.85	0.8
	0.59	0.62	0.6	0.6	0.61	0.6	0.61	0.63	0.56
7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.6
	0.45	0.4	0.4	0.4	0.45	0.45	0.35	0.4	0.4
	0.55	0.55	0.6	0.5	0.55	0.55	0.6	0.55	0.55
	0.7	0.6	0.5	0.65	0.65	0.55	0.65	0.6	0.6
	0.85	0.75	0.75	0.8	0.85	0.85	0.8	0.8	0.8
	0.5	0.7	0.75	0.65	0.55	0.65	0.65	0.65	0.75
	0.61	0.61	0.61	0.61	0.61	0.61	0.61	0.61	0.63
8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.55	0.6	0.55	0.5	0.6	0.5	0.5	0.55
	0.45	0.45	0.4	0.45	0.5	0.45	0.35	0.55	0.4
	0.55	0.55	0.55	0.6	0.55	0.6	0.55	0.45	0.5
	0.7	0.55	0.5	0.55	0.7	0.6	0.6	0.65	0.7
	0.85	0.75	0.7	0.85	0.85	0.85	0.75	0.85	0.8
	0.45	0.7	0.75	0.6	0.5	0.6	0.8	0.6	0.6
	0.2	0.15	0.25	0.15	0.15	0.15	0.1	0.1	0.15
	0.56	0.55	0.56	0.56	0.56	0.57	0.54	0.55	0.55
9	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.55	0.55	0.55	0.55	0.55	0.5	0.55
	0.4	0.4	0.45	0.45	0.4	0.4	0.4	0.5	0.4
	0.55	0.5	0.55	0.55	0.5	0.55	0.5	0.55	0.55
	0.75	0.65	0.65	0.55	0.7	0.6	0.65	0.65	0.6
	0.8	0.8	0.85	0.8	0.8	0.8	0.75	0.75	0.8
	0.45	0.65	0.55	0.7	0.6	0.7	0.7	0.6	0.65
	0.25	0.15	0.15	0.2	0.15	0.15	0.15	0.2	0.15

	0.15	0.15	0.1	0.15	0.1	0.2	0.15	0.1	0.15
	0.51	0.51	0.51	0.52	0.5	0.52	0.51	0.51	0.51
10	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.55	0.55	0.6	0.55	0.55	0.5	0.55	0.55
	0.45	0.4	0.4	0.4	0.35	0.45	0.5	0.4	0.45
	0.6	0.5	0.5	0.5	0.55	0.6	0.5	0.6	0.55
	0.55	0.65	0.7	0.65	0.55	0.6	0.7	0.65	0.65
	0.85	0.8	0.75	0.8	0.75	0.85	0.85	0.8	0.8
	0.6	0.65	0.6	0.65	0.8	0.55	0.5	0.6	0.65
	0.15	0.1	0.15	0.15	0.2	0.15	0.15	0.15	0.1
	0.1	0.3	0.25	0.15	0.15	0.2	0.2	0.2	0.1
	0.25	0.2	0.3	0.25	0.25	0.2	0.3	0.15	0.2
	0.48	0.49	0.49	0.49	0.49	0.49	0.49	0.48	0.48

Table 10.5: Comparison original vectors on different KNN with penalizing Multi-Word Existing Words methods

Iteration	1	2	3	4	5	6	7	8	9
1	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
2	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.55	0.5	0.6	0.55	0.5	0.55	0.55	0.55
	0.63	0.63	0.6	0.65	0.63	0.6	0.63	0.63	0.63
3	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.55	0.55	0.55	0.55	0.6	0.55	0.55	0.55
	0.45	0.45	0.45	0.4	0.4	0.4	0.45	0.4	0.4
	0.57	0.57	0.57	0.55	0.55	0.57	0.57	0.55	0.55
4	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.5	0.5	0.6	0.55	0.55	0.55	0.55	0.55	0.55
	0.45	0.6	0.35	0.45	0.35	0.4	0.4	0.4	0.4
	0.55	0.45	0.5	0.5	0.55	0.5	0.55	0.55	0.5
	0.55	0.56	0.54	0.55	0.54	0.54	0.55	0.55	0.54
5	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.55	0.55	0.55	0.55	0.55	0.6	0.55	0.55
	0.45	0.45	0.35	0.45	0.4	0.45	0.4	0.45	0.45
	0.55	0.55	0.6	0.55	0.5	0.55	0.55	0.55	0.55
	0.6	0.7	0.6	0.65	0.6	0.6	0.55	0.65	0.7
	0.57	0.59	0.56	0.58	0.55	0.57	0.56	0.58	0.59
6	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.6	0.55	0.55	0.6	0.5	0.55	0.55	0.5	0.6
	0.35	0.4	0.4	0.35	0.5	0.45	0.45	0.4	0.4
	0.55	0.55	0.5	0.5	0.5	0.55	0.55	0.5	0.55

	0.65	0.6	0.75	0.35	0.75	0.65	0.65	0.7	0.65
	0.75	0.75	0.8	0.7	0.8	0.8	0.85	0.8	0.85
	0.6	0.59	0.62	0.53	0.63	0.62	0.63	0.6	0.63
7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.55	0.6	0.55	0.55	0.55	0.55	0.55
	0.45	0.4	0.5	0.35	0.4	0.45	0.35	0.4	0.45
	0.6	0.55	0.5	0.5	0.55	0.55	0.55	0.55	0.55
	0.5	0.55	0.6	0.7	0.6	0.6	0.6	0.6	0.6
	0.85	0.85	0.8	0.75	0.8	0.85	0.75	0.8	0.8
	0.65	0.65	0.6	0.7	0.65	0.6	0.75	0.65	0.65
	0.61	0.61	0.61	0.61	0.61	0.61	0.61	0.61	0.61
8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.5	0.55	0.55	0.45	0.5	0.55	0.55	0.55
	0.45	0.45	0.4	0.4	0.6	0.6	0.45	0.4	0.4
	0.6	0.5	0.55	0.45	0.55	0.45	0.55	0.55	0.5
	0.5	0.65	0.6	0.7	0.6	0.5	0.6	0.55	0.7
	0.85	0.85	0.8	0.75	0.85	0.85	0.85	0.8	0.8
	0.6	0.6	0.7	0.6	0.65	0.65	0.6	0.75	0.6
	0.2	0.15	0.1	0.15	0.1	0.15	0.1	0.1	0.15
	0.56	0.55	0.55	0.54	0.56	0.55	0.55	0.55	0.55
9	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.5	0.6	0.55	0.55	0.55	0.55	0.55	0.55
	0.4	0.45	0.4	0.45	0.45	0.35	0.45	0.45	0.5
	0.5	0.55	0.55	0.55	0.5	0.55	0.55	0.5	0.5
	0.65	0.7	0.55	0.6	0.65	0.55	0.55	0.65	0.7
	0.8	0.85	0.85	0.85	0.85	0.75	0.85	0.85	0.85
	0.65	0.55	0.7	0.55	0.6	0.85	0.65	0.55	0.5
	0.15	0.15	0.15	0.2	0.15	0.15	0.15	0.15	0.25

	0.15	0.1	0.15	0.1	0.15	0.15	0.15	0.1	0.2
	0.51	0.51	0.52	0.51	0.51	0.51	0.51	0.5	0.53
10	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.55	0.55	0.55	0.6	0.55	0.55	0.5	0.55
	0.45	0.4	0.5	0.45	0.4	0.45	0.45	0.5	0.45
	0.6	0.55	0.45	0.6	0.55	0.5	0.55	0.5	0.5
	0.55	0.65	0.6	0.6	0.55	0.65	0.65	0.65	0.6
	0.8	0.8	0.8	0.85	0.75	0.8	0.85	0.85	0.85
	0.65	0.6	0.55	0.6	0.7	0.6	0.5	0.55	0.7
	0.15	0.15	0.25	0.1	0.2	0.15	0.2	0.15	0.1
	0.15	0.25	0.15	0.2	0.2	0.2	0.15	0.15	0.1
	0.25	0.3	0.15	0.15	0.2	0.25	0.25	0.2	0.3
	0.49	0.5	0.47	0.48	0.49	0.49	0.49	0.48	0.49

Table 10.7: Comparison Thesis

Iterations	cosine	cosElIn	jaccard	penEx	penMW	penMultiEx	knn - 7	penMultiExKnn - 9
1	0.65	0.85	0.25	0.85	0.75	0.9	0.65	0.4
	0.65	0.85	0.25	0.85	0.75	0.9	0.65	0.4
2	0.65	0.85	0.25	0.85	0.75	0.9	0.65	0.4
	0.5	0.6	0.3	0.55	0.5	0.55	0.55	0.8
	0.58	0.73	0.28	0.7	0.63	0.73	0.6	0.6
3	0.65	0.85	0.25	0.85	0.75	0.9	0.65	0.4
	0.5	0.6	0.3	0.55	0.5	0.55	0.55	0.8
	0.45	0.45	0.75	0.5	0.5	0.35	0.3	0.35
	0.53	0.63	0.43	0.63	0.58	0.6	0.5	0.52
4	0.65	0.85	0.25	0.85	0.75	0.9	0.65	0.4
	0.5	0.6	0.3	0.55	0.5	0.55	0.55	0.8
	0.45	0.45	0.75	0.5	0.5	0.35	0.3	0.35
	0.55	0.4	0.7	0.35	0.25	0.15	0.45	0.25
	0.54	0.58	0.5	0.56	0.5	0.49	0.49	0.45
5	0.65	0.85	0.25	0.85	0.75	0.9	0.65	0.4
	0.5	0.6	0.3	0.55	0.5	0.55	0.55	0.8
	0.45	0.45	0.75	0.5	0.5	0.35	0.3	0.35
	0.55	0.4	0.7	0.35	0.25	0.15	0.45	0.25
	0.45	0.2	0.3	0.2	0.2	0	0.3	0.3
	0.52	0.5	0.46	0.49	0.44	0.39	0.45	0.42
6	0.65	0.85	0.25	0.85	0.75	0.9	0.65	0.4
	0.5	0.6	0.3	0.55	0.5	0.55	0.55	0.8
	0.45	0.45	0.75	0.5	0.5	0.35	0.3	0.35
	0.55	0.4	0.7	0.35	0.25	0.15	0.45	0.25

	0.45	0.2	0.3	0.2	0.2	0	0.3	0.3
	0.25	0.25	0.5	0.3	0.15	0	0.2	0.15
	0.48	0.46	0.47	0.46	0.39	0.33	0.41	0.38
7	0.65	0.85	0.25	0.85	0.75	0.9	0.65	0.4
	0.5	0.6	0.3	0.55	0.5	0.55	0.55	0.8
	0.45	0.45	0.75	0.5	0.5	0.35	0.3	0.35
	0.55	0.4	0.7	0.35	0.25	0.15	0.45	0.25
	0.45	0.2	0.3	0.2	0.2	0	0.3	0.3
	0.25	0.25	0.5	0.3	0.15	0	0.2	0.15
	0.1	$5 \cdot 10^{-2}$	0.2	0.1	0.1	0.1	0.1	0.1
	0.42	0.4	0.43	0.41	0.35	0.29	0.36	0.34
8	0.65	0.85	0.25	0.85	0.75	0.9	0.65	0.4
	0.5	0.6	0.3	0.55	0.5	0.55	0.55	0.8
	0.45	0.45	0.75	0.5	0.5	0.35	0.3	0.35
	0.55	0.4	0.7	0.35	0.25	0.15	0.45	0.25
	0.45	0.2	0.3	0.2	0.2	0	0.3	0.3
	0.25	0.25	0.5	0.3	0.15	0	0.2	0.15
	0.1	$5 \cdot 10^{-2}$	0.2	0.1	0.1	0.1	0.1	0.1
	0.3	0.15	$5 \cdot 10^{-2}$	0.1	0.15	0.1	0.3	0
	0.41	0.37	0.38	0.37	0.33	0.27	0.36	0.29
9	0.65	0.85	0.25	0.85	0.75	0.9	0.65	0.4
	0.5	0.6	0.3	0.55	0.5	0.55	0.55	0.8
	0.45	0.45	0.75	0.5	0.5	0.35	0.3	0.35
	0.55	0.4	0.7	0.35	0.25	0.15	0.45	0.25
	0.45	0.2	0.3	0.2	0.2	0	0.3	0.3
	0.25	0.25	0.5	0.3	0.15	0	0.2	0.15
	0.1	$5 \cdot 10^{-2}$	0.2	0.1	0.1	0.1	0.1	0.1
	0.3	0.15	$5 \cdot 10^{-2}$	0.1	0.15	0.1	0.3	0

	0.1	0.2	0.1	0.2	0.2	$5 \cdot 10^{-2}$	0.1	0.1
	0.37	0.35	0.35	0.35	0.31	0.24	0.33	0.27

Table 10.9: Comparison Thesis on different KNN

Iteration	1	2	3	4	5	6	7	8	9
1	0.75	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
	0.75	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
2	0.75	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
	0.45	0.55	0.45	0.5	0.55	0.55	0.55	0.55	0.55
	0.6	0.6	0.55	0.58	0.6	0.6	0.6	0.6	0.6
3	0.75	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
	0.45	0.55	0.45	0.5	0.55	0.55	0.55	0.55	0.55
	0.45	0.45	0.45	0.4	0.3	0.3	0.3	0.3	0.35
	0.55	0.55	0.52	0.52	0.5	0.5	0.5	0.5	0.52
4	0.75	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
	0.45	0.55	0.45	0.5	0.55	0.55	0.55	0.55	0.55
	0.45	0.45	0.45	0.4	0.3	0.3	0.3	0.3	0.35
	0.25	0.15	0.4	0.4	0.4	0.45	0.45	0.55	0.5
	0.48	0.45	0.49	0.49	0.48	0.49	0.49	0.51	0.51
5	0.75	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
	0.45	0.55	0.45	0.5	0.55	0.55	0.55	0.55	0.55
	0.45	0.45	0.45	0.4	0.3	0.3	0.3	0.3	0.35
	0.25	0.15	0.4	0.4	0.4	0.45	0.45	0.55	0.5
	0.25	0.4	0.3	0.35	0.4	0.3	0.3	0.2	0.25
	0.43	0.44	0.45	0.46	0.46	0.45	0.45	0.45	0.46
6	0.75	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
	0.45	0.55	0.45	0.5	0.55	0.55	0.55	0.55	0.55
	0.45	0.45	0.45	0.4	0.3	0.3	0.3	0.3	0.35
	0.25	0.15	0.4	0.4	0.4	0.45	0.45	0.55	0.5

	0.25	0.4	0.3	0.35	0.4	0.3	0.3	0.2	0.25
	0.1	0.2	0.2	0.2	0.15	0.2	0.2	0.25	0.25
	0.38	0.4	0.41	0.42	0.41	0.41	0.41	0.42	0.43
7	0.75	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
	0.45	0.55	0.45	0.5	0.55	0.55	0.55	0.55	0.55
	0.45	0.45	0.45	0.4	0.3	0.3	0.3	0.3	0.35
	0.25	0.15	0.4	0.4	0.4	0.45	0.45	0.55	0.5
	0.25	0.4	0.3	0.35	0.4	0.3	0.3	0.2	0.25
	0.1	0.2	0.2	0.2	0.15	0.2	0.2	0.25	0.25
	0.3	0.2	0.25	$5 \cdot 10^{-2}$	0.15	0.1	0.1	0.1	0.1
	0.36	0.37	0.39	0.36	0.37	0.36	0.36	0.37	0.38
8	0.75	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
	0.45	0.55	0.45	0.5	0.55	0.55	0.55	0.55	0.55
	0.45	0.45	0.45	0.4	0.3	0.3	0.3	0.3	0.35
	0.25	0.15	0.4	0.4	0.4	0.45	0.45	0.55	0.5
	0.25	0.4	0.3	0.35	0.4	0.3	0.3	0.2	0.25
	0.1	0.2	0.2	0.2	0.15	0.2	0.2	0.25	0.25
	0.3	0.2	0.25	$5 \cdot 10^{-2}$	0.15	0.1	0.1	0.1	0.1
	0.1	0.2	0	0.2	0.1	0.25	0.3	0.25	0.15
	0.33	0.35	0.34	0.34	0.34	0.35	0.36	0.36	0.35
9	0.75	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
	0.45	0.55	0.45	0.5	0.55	0.55	0.55	0.55	0.55
	0.45	0.45	0.45	0.4	0.3	0.3	0.3	0.3	0.35
	0.25	0.15	0.4	0.4	0.4	0.45	0.45	0.55	0.5
	0.25	0.4	0.3	0.35	0.4	0.3	0.3	0.2	0.25
	0.1	0.2	0.2	0.2	0.15	0.2	0.2	0.25	0.25
	0.3	0.2	0.25	$5 \cdot 10^{-2}$	0.15	0.1	0.1	0.1	0.1
	0.1	0.2	0	0.2	0.1	0.25	0.3	0.25	0.15

	0.1	0	0.15	0.15	0.2	0.1	0.1	0.1	0.2
	0.31	0.31	0.32	0.32	0.32	0.32	0.33	0.33	0.33
10									

Table 10.11: Comparison Thesis on different KNN with penalizing Multi-Word Existing Words methods

Iteration	1	2	3	4	5	6	7	8	9
1	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
2	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.7	0.75	0.75	0.8	0.8	0.75	0.8	0.8	0.8
	0.55	0.58	0.58	0.6	0.6	0.58	0.6	0.6	0.6
3	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.7	0.75	0.75	0.8	0.8	0.75	0.8	0.8	0.8
	0.3	0.3	0.3	0.3	0.3	0.35	0.3	0.3	0.35
	0.47	0.48	0.48	0.5	0.5	0.5	0.5	0.5	0.52
4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.7	0.75	0.75	0.8	0.8	0.75	0.8	0.8	0.8
	0.3	0.3	0.3	0.3	0.3	0.35	0.3	0.3	0.35
	0.4	0.35	0.35	0.3	0.3	0.3	0.3	0.3	0.25
	0.45	0.45	0.45	0.45	0.45	0.45	0.45	0.45	0.45
5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.7	0.75	0.75	0.8	0.8	0.75	0.8	0.8	0.8
	0.3	0.3	0.3	0.3	0.3	0.35	0.3	0.3	0.35
	0.4	0.35	0.35	0.3	0.3	0.3	0.3	0.3	0.25
	0.15	0.15	0.15	0.2	0.2	0.25	0.25	0.3	0.3
	0.39	0.39	0.39	0.4	0.4	0.41	0.41	0.42	0.42
6	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.7	0.75	0.75	0.8	0.8	0.75	0.8	0.8	0.8
	0.3	0.3	0.3	0.3	0.3	0.35	0.3	0.3	0.35
	0.4	0.35	0.35	0.3	0.3	0.3	0.3	0.3	0.25

	0.15	0.15	0.15	0.2	0.2	0.25	0.25	0.3	0.3
	0.1	0.15	0.2	0.2	0.15	0.15	0.2	0.15	0.15
	0.34	0.35	0.36	0.37	0.36	0.37	0.38	0.38	0.38
7	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.7	0.75	0.75	0.8	0.8	0.75	0.8	0.8	0.8
	0.3	0.3	0.3	0.3	0.3	0.35	0.3	0.3	0.35
	0.4	0.35	0.35	0.3	0.3	0.3	0.3	0.3	0.25
	0.15	0.15	0.15	0.2	0.2	0.25	0.25	0.3	0.3
	0.1	0.15	0.2	0.2	0.15	0.15	0.2	0.15	0.15
	0.1	0.15	0.15	0.15	0.2	0.15	0.1	0.1	0.1
	0.31	0.32	0.33	0.34	0.34	0.34	0.34	0.34	0.34
8	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.7	0.75	0.75	0.8	0.8	0.75	0.8	0.8	0.8
	0.3	0.3	0.3	0.3	0.3	0.35	0.3	0.3	0.35
	0.4	0.35	0.35	0.3	0.3	0.3	0.3	0.3	0.25
	0.15	0.15	0.15	0.2	0.2	0.25	0.25	0.3	0.3
	0.1	0.15	0.2	0.2	0.15	0.15	0.2	0.15	0.15
	0.1	0.15	0.15	0.15	0.2	0.15	0.1	0.1	0.1
	0.15	0.1	$5 \cdot 10^{-2}$	0	0	0	0	0	0
	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.29
9	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.7	0.75	0.75	0.8	0.8	0.75	0.8	0.8	0.8
	0.3	0.3	0.3	0.3	0.3	0.35	0.3	0.3	0.35
	0.4	0.35	0.35	0.3	0.3	0.3	0.3	0.3	0.25
	0.15	0.15	0.15	0.2	0.2	0.25	0.25	0.3	0.3
	0.1	0.15	0.2	0.2	0.15	0.15	0.2	0.15	0.15
	0.1	0.15	0.15	0.15	0.2	0.15	0.1	0.1	0.1
	0.15	0.1	$5 \cdot 10^{-2}$	0	0	0	0	0	0

	$5 \cdot 10^{-2}$	0	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	0.1	0.1	0.1	0.1	0.1
	0.26	0.26	0.27	0.27	0.27	0.27	0.27	0.27	0.27
10									

Table 10.13: Comparison Thesis without quantifiers

Iterations	cosine	cosElIn	jaccard	penEx	penMW	penMultiEx	knn - 8	penMultiExKnn - 8
1	0.8	0.9	0.25	0.9	0.7	0.9	0.75	0.4
	0.8	0.9	0.25	0.9	0.7	0.9	0.75	0.4
2	0.8	0.9	0.25	0.9	0.7	0.9	0.75	0.4
	0.7	0.55	0.3	0.55	0.6	0.55	0.7	0.8
	0.75	0.73	0.28	0.73	0.65	0.73	0.73	0.6
3	0.8	0.9	0.25	0.9	0.7	0.9	0.75	0.4
	0.7	0.55	0.3	0.55	0.6	0.55	0.7	0.8
	0.55	0.5	0.65	0.45	0.55	0.4	0.6	0.35
	0.68	0.65	0.4	0.63	0.62	0.62	0.68	0.52
4	0.8	0.9	0.25	0.9	0.7	0.9	0.75	0.4
	0.7	0.55	0.3	0.55	0.6	0.55	0.7	0.8
	0.55	0.5	0.65	0.45	0.55	0.4	0.6	0.3
	0.55	0.65	0.7	0.65	0.3	0.15	0.35	0.35
	0.65	0.65	0.48	0.64	0.54	0.5	0.6	0.48
5	0.8	0.9	0.25	0.9	0.7	0.9	0.75	0.4
	0.7	0.55	0.3	0.55	0.6	0.55	0.7	0.8
	0.55	0.5	0.65	0.45	0.55	0.4	0.6	0.35
	0.55	0.65	0.7	0.65	0.3	0.15	0.35	0.35
	0.35	0.4	0.45	0.3	0.15	0	0.25	0.2
	0.59	0.6	0.47	0.57	0.46	0.4	0.53	0.42
6	0.8	0.9	0.25	0.9	0.7	0.9	0.75	0.4
	0.7	0.55	0.3	0.55	0.6	0.55	0.7	0.8
	0.55	0.5	0.65	0.45	0.55	0.4	0.6	0.35
	0.55	0.65	0.7	0.65	0.3	0.15	0.35	0.35

	0.35	0.4	0.45	0.3	0.15	0	0.25	0.2
	0.3	0.3	0.45	0.4	0.1	0	0.3	0.15
	0.54	0.55	0.47	0.54	0.4	0.33	0.49	0.38
7	0.8	0.9	0.25	0.9	0.7	0.9	0.75	0.4
	0.7	0.55	0.3	0.55	0.6	0.55	0.7	0.8
	0.55	0.5	0.65	0.45	0.55	0.4	0.6	0.35
	0.55	0.65	0.7	0.65	0.3	0.15	0.35	0.35
	0.35	0.4	0.45	0.3	0.15	0	0.25	0.2
	0.3	0.3	0.45	0.4	0.1	0	0.3	0.15
	0.4	0.15	0.2	$5 \cdot 10^{-2}$	0.1	0	0.3	0.15
	0.52	0.49	0.43	0.47	0.36	0.29	0.46	0.34
8	0.8	0.9	0.25	0.9	0.7	0.9	0.75	0.4
	0.7	0.55	0.3	0.55	0.6	0.55	0.7	0.8
	0.55	0.5	0.65	0.45	0.55	0.4	0.6	0.35
	0.55	0.65	0.7	0.65	0.3	0.15	0.35	0.35
	0.35	0.4	0.45	0.3	0.15	0	0.25	0.2
	0.3	0.3	0.45	0.4	0.1	0	0.3	0.15
	0.4	0.15	0.2	$5 \cdot 10^{-2}$	0.1	0	0.3	0.15
	0.25	0.2	0.15	0.15	0.15	$5 \cdot 10^{-2}$	0.25	0
	0.49	0.46	0.39	0.43	0.33	0.26	0.44	0.3
9	0.8	0.9	0.25	0.9	0.7	0.9	0.75	0.4
	0.7	0.55	0.3	0.55	0.6	0.55	0.7	0.8
	0.55	0.5	0.65	0.45	0.55	0.4	0.6	0.35
	0.55	0.65	0.7	0.65	0.3	0.15	0.35	0.35
	0.35	0.4	0.45	0.3	0.15	0	0.25	0.2
	0.3	0.3	0.45	0.4	0.1	0	0.3	0.15
	0.4	0.15	0.2	$5 \cdot 10^{-2}$	0.1	0	0.3	0.15
	0.25	0.2	0.15	0.15	0.15	$5 \cdot 10^{-2}$	0.25	0

	0.3	0.15	$5 \cdot 10^{-2}$	0.15	$5 \cdot 10^{-2}$	0	0.4	$5 \cdot 10^{-2}$
	0.47	0.42	0.36	0.4	0.3	0.23	0.43	0.27
10	0.8	0.9	0.25	0.9	0.7	0.9	0.75	0.4
	0.7	0.55	0.3	0.55	0.6	0.55	0.7	0.8
	0.55	0.5	0.65	0.45	0.55	0.4	0.6	0.35
	0.55	0.65	0.7	0.65	0.3	0.15	0.35	0.35
	0.35	0.4	0.45	0.3	0.15	0	0.25	0.2
	0.3	0.3	0.45	0.4	0.1	0	0.3	0.15
	0.4	0.15	0.2	$5 \cdot 10^{-2}$	0.1	0	0.3	0.15
	0.25	0.2	0.15	0.15	0.15	$5 \cdot 10^{-2}$	0.25	0
	0.3	0.15	$5 \cdot 10^{-2}$	0.15	$5 \cdot 10^{-2}$	0	0.4	$5 \cdot 10^{-2}$
	0.25	0.2	0	0.2	$5 \cdot 10^{-2}$	0.1	0.25	$5 \cdot 10^{-2}$
	0.45	0.4	0.32	0.38	0.28	0.22	0.42	0.25

Table 10.15: Comparison Thesis without quantifiers on different KNN

Iteration	1	2	3	4	5	6	7	8	9
1	0.9	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	0.9	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
2	0.9	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	0.6	0.65	0.75	0.75	0.7	0.7	0.7	0.7	0.7
	0.75	0.7	0.75	0.75	0.73	0.73	0.73	0.73	0.73
3	0.9	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	0.6	0.65	0.75	0.75	0.7	0.7	0.7	0.7	0.7
	0.35	0.4	0.4	0.5	0.55	0.55	0.6	0.6	0.55
	0.62	0.6	0.63	0.67	0.67	0.67	0.68	0.68	0.67
4	0.9	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	0.6	0.65	0.75	0.75	0.7	0.7	0.7	0.7	0.7
	0.35	0.4	0.4	0.5	0.55	0.55	20	0.6	0.6
	0.15	0.4	0.5	0.35	0.35	0.35	0.35	0.35	0.55
	0.5	0.55	0.6	0.59	0.59	0.59	0.6	0.6	0.64
5	0.9	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	0.6	0.65	0.75	0.75	0.7	0.7	0.7	0.7	0.7
	0.35	0.4	0.4	0.5	0.55	0.55	0.6	0.6	0.55
	0.15	0.4	0.5	0.35	0.35	0.35	0.35	0.35	0.55
	$5 \cdot 10^{-2}$	0.25	$5 \cdot 10^{-2}$	0.2	0.2	0.25	0.2	0.25	0.25
	0.41	0.49	0.49	0.51	0.51	0.52	0.52	0.53	0.56
6	0.9	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	0.6	0.65	0.75	0.75	0.7	0.7	20	0.7	0.7
	0.35	0.4	0.4	0.5	0.55	0.55	0.6	0.6	0.55
	0.15	0.4	0.5	0.35	0.35	0.35	0.35	0.35	0.55

	$5 \cdot 10^{-2}$	0.25	$5 \cdot 10^{-2}$	0.2	0.2	0.25	0.2	0.25	0.25
	0.35	0.25	0.45	0.35	0.35	0.35	0.35	0.3	0.2
	0.4	0.45	0.48	0.48	0.48	0.49	0.49	0.49	0.5
7	0.9	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	0.6	0.65	0.75	0.75	0.7	0.7	0.7	0.7	0.7
	0.35	0.4	0.4	0.5	0.55	0.55	0.6	0.6	0.55
	0.15	0.4	0.5	0.35	0.35	0.35	0.35	0.35	0.55
	$5 \cdot 10^{-2}$	0.25	$5 \cdot 10^{-2}$	0.2	0.2	0.25	0.2	0.25	0.25
	0.35	0.25	0.45	0.35	0.35	0.35	0.35	0.3	0.2
	0.2	0.45	0.25	0.35	0.35	0.3	0.35	0.3	0.25
	0.37	0.45	0.45	0.46	0.46	0.46	0.47	0.46	0.46
8	0.9	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	0.6	0.65	0.75	0.75	0.7	0.7	0.7	0.7	0.7
	0.35	0.4	0.4	0.5	0.55	0.55	0.6	0.6	0.55
	0.15	0.4	0.5	0.35	0.35	0.35	0.35	0.35	0.55
	$5 \cdot 10^{-2}$	0.25	$5 \cdot 10^{-2}$	0.2	0.2	0.25	0.2	0.25	0.25
	0.35	0.25	0.45	0.35	0.35	0.35	0.35	0.3	0.2
	0.2	0.45	0.25	0.35	0.35	0.3	0.35	0.3	0.25
	0.15	0.1	0.2	0.1	0.1	0.2	0.15	0.25	0.25
	0.34	0.41	0.42	0.42	0.42	0.43	0.43	0.44	0.44
9	0.9	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	0.6	0.65	0.75	0.75	0.7	0.7	0.7	0.7	0.7
	0.35	0.4	0.4	0.5	0.55	0.55	0.6	0.6	0.55
	0.15	0.4	0.5	0.35	0.35	0.35	0.35	0.35	0.55
	$5 \cdot 10^{-2}$	0.25	$5 \cdot 10^{-2}$	0.2	0.2	0.25	0.2	0.25	0.25
	0.35	0.25	0.45	0.35	0.35	0.35	0.35	0.3	0.2
	0.2	0.45	0.25	0.35	0.35	0.3	0.35	0.3	0.25
	0.15	0.1	0.2	0.1	0.1	0.2	0.15	0.25	0.25

	0.2	0.1	0.1	0.15	0.15	0.25	0.3	0.4	0.45
	0.33	0.37	0.38	0.39	0.39	0.41	0.42	0.43	0.44
10	0.9	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	0.6	0.65	0.75	0.75	0.7	0.7	0.7	0.7	0.7
	0.35	0.4	0.4	0.5	0.55	0.55	0.6	0.6	0.55
	0.15	0.4	0.5	0.35	0.35	0.35	0.35	0.35	0.55
	$5 \cdot 10^{-2}$	0.25	$5 \cdot 10^{-2}$	0.2	0.2	0.25	0.2	0.25	0.25
	0.35	0.25	0.45	0.35	0.35	0.35	0.35	0.3	0.2
	0.2	0.45	0.25	0.35	0.35	0.3	0.35	0.3	0.25
	0.15	0.1	0.2	0.1	0.1	0.2	0.15	0.25	0.25
	0.2	0.1	0.1	0.15	0.15	0.25	0.3	0.4	0.45
	0.15	0.1	0.1	0.25	0.35	0.3	0.3	0.25	0.2
	0.31	0.35	0.36	0.38	0.39	0.4	0.41	0.42	0.42

Table 10.17: Comparison Thesis without quantifiers on different KNN with penalizing Multi-Word Existing Words methods

Iteration	1	2	3	4	5	6	7	8	9
1	0.35	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.35	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
2	0.35	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.8	0.85	0.8	0.8	0.8	0.8	0.8	0.8	0.8
	0.58	0.63	0.6	0.6	0.6	0.6	0.6	0.6	0.6
3	0.35	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.8	0.85	0.8	0.8	0.8	0.8	0.8	0.8	0.8
	0.25	0.25	0.3	0.3	0.3	0.3	0.3	0.35	0.35
	0.47	0.5	0.5	0.5	0.5	0.5	0.5	0.52	0.52
4	0.35	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.8	0.85	0.8	0.8	0.8	0.8	0.8	0.8	0.8
	0.55	0.25	0.25	0.3	0.3	0.3	0.3	0.3	0.35
	0.35	0.3	0.3	0.3	0.3	0.3	0.35	0.35	0.35
	0.44	0.45	0.45	0.45	0.45	0.45	0.46	0.48	0.48
5	0.35	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.8	0.85	0.8	0.8	0.8	0.8	0.8	0.8	0.8
	0.25	0.25	0.3	0.3	0.3	0.3	0.3	0.35	0.35
	0.35	0.3	0.3	0.3	0.3	0.3	0.35	0.35	0.35
	0.1	0.15	0.15	0.25	0.2	0.25	0.15	0.2	0.15
	0.37	0.39	0.39	0.41	0.4	0.41	0.4	0.42	0.41
6	0.35	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.7	0.8	0.85	0.8	0.8	0.8	0.8	0.8	0.8
	0.25	0.25	0.3	0.3	0.3	0.3	0.3	0.35	0.35

	0.35	0.3	0.3	0.3	0.3	0.3	0.35	0.35	0.35
	0.1	0.15	0.15	0.25	0.2	0.25	0.15	0.2	0.15
	0.2	0.2	0.2	0.2	0.2	0.15	0.2	0.15	0.15
	0.34	0.36	0.36	0.38	0.37	0.37	0.37	0.38	0.37
7	0.35	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.8	0.85	0.8	0.8	0.8	0.8	0.8	0.8	0.8
	0.25	0.25	0.3	0.3	0.3	0.3	0.3	0.35	0.35
	0.35	0.3	0.3	0.3	0.3	0.3	0.35	0.35	0.35
	0.1	0.15	0.15	0.25	0.2	0.25	0.15	0.2	0.15
	0.2	0.2	0.2	0.2	0.2	0.15	0.2	0.15	0.15
	0.1	0.1	0.2	$5 \cdot 10^{-2}$	0.1	0.15	0.2	0.15	0.2
	0.31	0.32	0.34	0.33	0.33	0.34	0.34	0.34	0.34
8	0.35	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.8	0.85	0.8	0.8	0.8	0.8	0.8	0.8	0.8
	0.25	0.25	0.3	0.3	0.3	0.3	0.3	0.35	0.35
	0.35	0.3	0.3	0.3	0.3	0.3	0.35	0.35	0.35
	0.1	0.15	0.15	0.25	0.2	0.25	0.15	0.2	0.15
	0.2	0.2	0.2	0.2	0.2	0.15	0.2	0.15	0.15
	0.1	0.1	0.2	$5 \cdot 10^{-2}$	0.1	0.15	0.2	0.15	0.2
	0.2	0.2	0.15	0.15	0.1	$5 \cdot 10^{-2}$	0	0	0
	0.29	0.31	0.31	0.31	0.3	0.3	0.3	0.3	0.3
9	0.35	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.8	0.85	0.8	0.8	0.8	0.8	0.8	0.8	0.8
	0.25	0.25	0.3	0.3	0.3	0.3	0.3	0.35	0.35
	0.35	0.3	0.3	0.3	0.3	0.3	0.35	0.35	0.35
	0.1	0.15	0.15	0.25	0.2	0.25	0.15	0.2	0.15
	0.2	0.2	0.2	0.2	0.2	0.15	0.2	0.15	0.15
	0.1	0.1	0.2	$5 \cdot 10^{-2}$	0.1	0.15	0.2	0.15	0.2

	0.2	0.2	0.15	0.15	0.1	$5 \cdot 10^{-2}$	0	0	0
	0.1	$5 \cdot 10^{-2}$	0	$5 \cdot 10^{-2}$	0.1	0.1	0.1	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.27	0.28	0.28	0.28	0.28	0.28	0.28	0.27	0.27
10	0.35	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	0.8	0.85	0.8	0.8	0.8	0.8	0.8	0.8	0.8
	0.25	0.25	0.3	0.3	0.3	0.3	0.3	0.35	0.35
	0.35	0.3	0.3	0.3	0.3	0.3	0.35	0.35	0.35
	0.1	0.15	0.15	0.25	0.2	0.25	0.15	0.2	0.15
	0.2	0.2	0.2	0.2	0.2	0.15	0.2	0.15	0.15
	0.1	0.1	0.2	$5 \cdot 10^{-2}$	0.1	0.15	0.2	0.15	0.2
	0.2	0.2	0.15	0.15	0.1	$5 \cdot 10^{-2}$	0	0	0
	0.1	$5 \cdot 10^{-2}$	0	$5 \cdot 10^{-2}$	0.1	0.1	0.1	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	$5 \cdot 10^{-2}$	0	$5 \cdot 10^{-2}$	0	0	0	0	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.25	0.25	0.26	0.25	0.25	0.25	0.25	0.25	0.25

Table 10.19: Comparison Google vectors

Iteration	cosine	cosElIn	jaccard	penEx	penMW	penMultiEx	knn - 8	penMultiExKnn - 6
1	0.75	0.85	0.2	0.85	0.7	0.85	0.7	$5 \cdot 10^{-2}$
	0.75	0.85	0.2	0.85	0.7	0.85	0.7	$5 \cdot 10^{-2}$
2	0.75	0.85	0.2	0.85	0.7	0.85	0.7	$5 \cdot 10^{-2}$
	0.55	0.55	0.3	0.6	0.7	0.6	0.7	0.55
	0.65	0.7	0.25	0.73	0.7	0.73	0.7	0.3
3	0.75	0.85	0.2	0.85	0.7	0.85	0.7	$5 \cdot 10^{-2}$
	0.55	0.55	0.3	0.6	0.7	0.6	0.7	0.55
	0.75	0.7	0.7	0.55	0.5	0.35	0.6	0.55
	0.68	0.7	0.4	0.67	0.63	0.6	0.67	0.38
4	0.75	0.85	0.2	0.85	0.7	0.85	0.7	$5 \cdot 10^{-2}$
	0.55	0.55	0.3	0.6	0.7	0.6	0.7	0.55
	0.75	0.7	0.7	0.55	0.5	0.35	0.6	0.55
	0.45	0.45	0.8	0.5	0.2	$5 \cdot 10^{-2}$	0.2	0.45
	0.63	0.64	0.5	0.63	0.53	0.46	0.55	0.4
5	0.75	0.85	0.2	0.85	0.7	0.85	0.7	$5 \cdot 10^{-2}$
	0.55	0.55	0.3	0.6	0.7	0.6	0.7	0.55
	0.75	0.7	0.7	0.55	0.5	0.35	0.6	0.55
	0.45	0.45	0.8	0.5	0.2	$5 \cdot 10^{-2}$	0.2	0.45
	0.3	0.35	0.3	0.4	0.1	0.15	0.35	0.2
	0.56	0.58	0.46	0.58	0.44	0.4	0.51	0.36
6	0.75	0.85	0.2	0.85	0.7	0.85	0.7	$5 \cdot 10^{-2}$
	0.55	0.55	0.3	0.6	0.7	0.6	0.7	0.55
	0.75	0.7	0.7	0.55	0.5	0.35	0.6	0.55
	0.45	0.45	0.8	0.5	0.2	$5 \cdot 10^{-2}$	0.25	0.45

	0.3	0.35	0.3	0.4	0.1	0.15	0.3	0.2
	0.3	0.2	0.3	$5 \cdot 10^{-2}$	0.1	0	0.3	0.25
	0.52	0.52	0.43	0.49	0.38	0.33	0.48	0.34
7	0.75	0.85	0.2	0.85	0.7	0.85	0.7	$5 \cdot 10^{-2}$
	0.55	0.55	0.3	0.6	0.7	0.6	0.7	0.55
	0.75	0.7	0.7	0.55	0.5	0.35	0.6	0.55
	0.45	0.45	0.8	0.5	0.2	$5 \cdot 10^{-2}$	0.2	0.45
	0.3	0.35	0.3	0.4	0.1	0.15	0.35	0.2
	0.3	0.2	0.3	$5 \cdot 10^{-2}$	0.1	0	0.3	0.25
	0.2	$5 \cdot 10^{-2}$	0.3	0.15	0.15	$5 \cdot 10^{-2}$	0.25	0.2
	0.47	0.45	0.41	0.44	0.35	0.29	0.44	0.32
8	0.75	0.85	0.2	0.85	0.7	0.85	0.7	$5 \cdot 10^{-2}$
	0.55	0.55	0.3	0.6	0.7	0.6	0.7	0.55
	0.75	0.7	0.7	0.55	0.5	0.35	0.6	0.55
	0.45	0.45	0.8	0.5	0.2	$5 \cdot 10^{-2}$	0.2	0.45
	0.3	0.35	0.3	0.4	0.1	0.15	0.35	0.2
	0.3	0.2	0.3	$5 \cdot 10^{-2}$	0.1	0	0.3	0.25
	0.2	$5 \cdot 10^{-2}$	0.3	0.15	0.15	$5 \cdot 10^{-2}$	0.25	0.2
	0.3	0.1	0.2	$5 \cdot 10^{-2}$	0.15	0.1	$5 \cdot 10^{-2}$	0.1
	0.45	0.41	0.39	0.39	0.33	0.27	0.39	0.29
9	0.75	0.85	0.2	0.85	0.7	0.85	0.7	$5 \cdot 10^{-2}$
	0.55	0.55	0.3	0.6	0.7	0.6	0.7	0.55
	0.75	0.7	0.7	0.55	0.5	0.35	0.6	0.55
	0.45	0.45	0.8	0.5	0.2	$5 \cdot 10^{-2}$	0.2	0.45
	0.3	0.35	0.3	0.4	0.1	0.15	0.35	0.2
	0.3	0.2	0.3	$5 \cdot 10^{-2}$	0.1	0	0.3	0.25
	0.2	$5 \cdot 10^{-2}$	0.3	0.15	0.15	$5 \cdot 10^{-2}$	0.25	0.2
	0.3	0.1	0.2	$5 \cdot 10^{-2}$	0.15	0.1	$5 \cdot 10^{-2}$	0.1

	0.25	0.15	0.1	$5 \cdot 10^{-2}$	0.1	0	0.15	0.15
	0.43	0.38	0.36	0.36	0.3	0.24	0.37	0.28
10	0.75	0.85	0.2	0.85	0.7	0.85	0.7	$5 \cdot 10^{-2}$
	0.55	0.55	0.3	0.6	0.7	0.6	0.7	0.55
	0.75	0.7	0.7	0.55	0.5	0.35	0.6	0.55
	0.45	0.45	0.8	0.5	0.2	$5 \cdot 10^{-2}$	0.2	0.45
	0.3	0.35	0.3	0.4	0.1	0.15	0.35	0.2
	0.3	0.2	0.3	$5 \cdot 10^{-2}$	0.1	0	0.3	0.25
	0.2	$5 \cdot 10^{-2}$	0.3	0.15	0.15	$5 \cdot 10^{-2}$	0.25	0.2
	0.3	0.1	0.2	$5 \cdot 10^{-2}$	0.15	0.1	$5 \cdot 10^{-2}$	0.1
	0.25	0.15	0.1	$5 \cdot 10^{-2}$	0.1	0	0.15	0.15
	0.15	0.2	0.1	0.15	0.3	0	$5 \cdot 10^{-2}$	0
	0.4	0.36	0.33	0.34	0.3	0.22	0.34	0.25

Table 10.21: Comparison Google vectors on different KNN

Iteration	1	2	3	4	5	6	7	8	9
1	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
2	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.6	0.65	0.65	0.65	0.65	0.7	0.7
	0.68	0.65	0.65	0.68	0.68	0.68	0.68	0.7	0.7
3	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.6	0.65	0.65	0.65	0.65	0.7	0.7
	0.25	0.3	0.45	0.45	0.5	0.55	0.6	0.6	0.5
	0.53	0.53	0.58	0.6	0.62	0.63	0.65	0.67	0.63
4	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.6	0.65	0.65	0.65	0.65	0.7	0.7
	0.25	0.3	0.45	0.45	0.5	0.55	0.6	0.6	0.5
	0.3	0.4	0.35	0.35	0.2	0.25	0.25	0.2	0.35
	0.48	0.5	0.53	0.54	0.51	0.54	0.55	0.55	0.56
5	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.6	0.65	0.65	0.65	0.65	0.7	0.7
	0.25	0.3	0.45	0.45	0.5	0.55	0.6	0.6	0.5
	0.3	0.4	0.35	0.35	0.2	0.25	0.25	0.2	0.35
	0.15	0.1	0.15	0.15	0.3	0.3	0.3	0.35	0.25
	0.41	0.42	0.45	0.46	0.47	0.49	0.5	0.51	0.5
6	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.6	0.65	0.65	0.65	0.65	0.7	0.7
	0.25	0.3	0.45	0.45	0.5	0.55	0.6	0.6	0.5
	0.3	0.4	0.35	0.35	0.2	0.25	0.25	0.25	0.35

	0.15	0.1	0.15	0.15	0.3	0.3	0.3	0.3	0.25
	$5 \cdot 10^{-2}$	0.2	0.15	0.2	0.3	0.25	0.3	0.3	0.35
	0.35	0.38	0.4	0.42	0.44	0.45	0.47	0.48	0.48
7	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.6	0.65	0.65	0.65	0.65	0.7	0.7
	0.25	0.3	0.45	0.45	0.5	0.55	0.6	0.6	0.5
	0.3	0.4	0.35	0.35	0.2	0.25	0.25	0.2	0.35
	0.15	0.1	0.15	0.15	0.3	0.3	0.3	0.35	0.25
	$5 \cdot 10^{-2}$	0.2	0.15	0.2	0.3	0.25	0.3	0.3	0.35
	0.25	0.2	0.15	0.2	0.15	0.25	0.2	0.25	0.25
	0.34	0.36	0.36	0.39	0.4	0.42	0.43	0.44	0.44
8	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.6	0.65	0.65	0.65	0.65	0.7	0.7
	0.25	0.3	0.45	0.45	0.5	0.55	0.6	0.6	0.5
	0.3	0.4	0.35	0.35	0.2	0.25	0.25	0.2	0.35
	0.15	0.1	0.15	0.15	0.3	0.3	0.3	0.35	0.25
	$5 \cdot 10^{-2}$	0.2	0.15	0.2	0.3	0.3	0.3	0.3	0.35
	0.25	0.2	0.15	0.2	0.15	0.2	0.2	0.25	0.25
	$5 \cdot 10^{-2}$	0.15	0.2	0.25	0.3	0.2	0.15	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.3	0.33	0.34	0.37	0.39	0.39	0.39	0.39	0.39
9	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.6	0.65	0.65	0.65	0.65	0.7	0.7
	0.25	0.3	0.45	0.45	0.5	0.55	0.6	0.6	0.5
	0.3	0.4	0.35	0.35	0.2	0.25	0.25	0.2	0.35
	0.15	0.1	0.15	0.15	0.3	0.3	0.3	0.35	0.25
	$5 \cdot 10^{-2}$	0.2	0.15	0.2	0.3	0.25	0.3	0.3	0.35
	0.25	0.2	0.15	0.2	0.15	0.25	0.2	0.25	0.25
	$5 \cdot 10^{-2}$	0.15	0.2	0.25	0.3	0.2	0.15	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$

	0.2	$5 \cdot 10^{-2}$	0.25	0.2	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	0.15	0.15	0.15
	0.29	0.3	0.33	0.35	0.35	0.36	0.37	0.37	0.37
10	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	0.55	0.6	0.6	0.65	0.65	0.65	0.65	0.7	0.7
	0.25	0.3	0.45	0.45	0.5	0.55	0.6	0.6	0.5
	0.3	0.4	0.35	0.35	0.2	0.25	0.25	0.2	0.35
	0.15	0.1	0.15	0.15	0.3	0.3	0.3	0.35	0.25
	$5 \cdot 10^{-2}$	0.2	0.15	0.2	0.3	0.25	0.3	0.3	0.35
	0.25	0.2	0.15	0.2	0.15	0.25	0.2	0.25	0.25
	$5 \cdot 10^{-2}$	0.15	0.2	0.25	0.3	0.2	0.15	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.2	$5 \cdot 10^{-2}$	0.25	0.2	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	0.15	0.15	0.15
	0.1	0.15	0.2	$5 \cdot 10^{-2}$	0.15	0.15	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.27	0.29	0.32	0.32	0.33	0.34	0.34	0.34	0.34

Table 10.23: Comparison Google vectors on different KNN with penalizing Multi-Word Existing Words methods

Iteration	1	2	3	4	5	6	7	8	9
1	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
2	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55
	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
3	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55
	0.55	0.5	0.45	0.5	0.5	0.55	0.6	0.6	0.6
	0.38	0.37	0.35	0.37	0.37	0.38	0.4	0.4	0.4
4	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55
	0.55	0.5	0.45	0.5	0.5	0.55	0.6	0.6	0.6
	0.3	0.45	0.55	0.5	0.45	0.45	0.4	0.4	0.45
	0.36	0.39	0.4	0.4	0.39	0.4	0.4	0.4	0.41
5	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55
	0.55	0.5	0.45	0.5	0.5	0.55	0.6	0.6	0.6
	0.3	0.45	0.55	0.5	0.45	0.45	0.4	0.4	0.45
	0.2	0.15	0.2	0.15	0.25	0.2	0.25	0.25	0.25
	0.33	0.34	0.36	0.35	0.36	0.36	0.37	0.37	0.38
6	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55
	0.55	0.5	0.45	0.5	0.5	0.55	0.6	0.6	0.6
	0.3	0.45	0.55	0.5	0.45	0.45	0.4	0.4	0.45

	0.2	0.15	0.2	0.15	0.25	0.2	0.25	0.25	0.25
	0.3	0.35	0.2	0.2	0.25	0.25	0.2	0.2	0.15
	0.33	0.34	0.33	0.33	0.34	0.34	0.34	0.34	0.34
7	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55
	0.55	0.5	0.45	0.5	0.5	0.55	0.6	0.6	0.6
	0.3	0.45	0.55	0.5	0.45	0.45	0.4	0.4	0.45
	0.2	0.15	0.2	0.15	0.25	0.2	0.25	0.25	0.25
	0.3	0.35	0.2	0.2	0.25	0.25	0.2	0.2	0.15
	0.2	0.2	0.3	0.3	0.2	0.2	0.2	0.2	0.25
	0.31	0.32	0.33	0.32	0.32	0.32	0.32	0.32	0.33
8	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55
	0.55	0.5	0.45	0.5	0.5	0.55	0.6	0.6	0.6
	0.3	0.45	0.55	0.5	0.45	0.45	0.4	0.4	0.45
	0.2	0.15	0.2	0.15	0.25	0.2	0.25	0.25	0.25
	0.3	0.35	0.2	0.2	0.25	0.25	0.2	0.2	0.15
	0.2	0.2	0.3	0.3	0.2	0.2	0.2	0.2	0.25
	0.2	0.15	0.1	0.1	0.1	0.1	0.1	0.1	$5 \cdot 10^{-2}$
	0.29	0.3	0.3	0.29	0.29	0.29	0.29	0.29	0.29
9	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55
	0.55	0.5	0.45	0.5	0.5	0.55	0.6	0.6	0.6
	0.3	0.45	0.55	0.5	0.45	0.45	0.4	0.4	0.45
	0.2	0.15	0.2	0.15	0.25	0.2	0.25	0.25	0.25
	0.3	0.35	0.2	0.2	0.25	0.25	0.2	0.2	0.15
	0.2	0.2	0.3	0.3	0.2	0.2	0.2	0.2	0.25
	0.2	0.15	0.1	0.1	0.1	0.1	0.1	0.1	$5 \cdot 10^{-2}$

	$5 \cdot 10^{-2}$	0.1	$5 \cdot 10^{-2}$	0.1	0.15	0.15	0.1	0.1	0.1
	0.27	0.28	0.27	0.27	0.28	0.28	0.27	0.27	0.27
10	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55
	0.55	0.5	0.45	0.5	0.5	0.55	0.6	0.6	0.6
	0.3	0.45	0.55	0.5	0.45	0.45	0.4	0.4	0.45
	0.2	0.15	0.2	0.15	0.25	0.2	0.25	0.25	0.25
	0.3	0.35	0.2	0.2	0.25	0.25	0.2	0.2	0.15
	0.2	0.2	0.3	0.3	0.2	0.2	0.2	0.2	0.25
	0.2	0.15	0.1	0.1	0.1	0.1	0.1	0.1	$5 \cdot 10^{-2}$
	$5 \cdot 10^{-2}$	0.1	$5 \cdot 10^{-2}$	0.1	0.15	0.15	0.1	0.1	0.1
	$5 \cdot 10^{-2}$	0	0	0	0	0	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$
	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25

Figure 10.3: Example news post from the DEA

```
February 22, 2007
Contact: Ruth Porter-Whipple
Number: 404-893-7128
Judge Hands Out Over 20 Year Sentence
FEB 27  -- COLUMBUS - On February 21, 2007, United States District
Court Judge Clay Land sentenced Torrance "Bookie" HILL, 31 years of
age, of Columbus, Georgia to a term of 294 months imprisonment for
HILL's involvement in a cocaine and marijuana distribution
organization.
Torrance HILL had earlier pled guilty to a four count information
alleging two counts of conspiracy to distribute more than five
kilograms of cocaine, one count of distribution of more than 50
grams of crack cocaine, and one count of possession with intent to
distribute more than 100 kilograms of marijuana.
HILL'S arrest and conviction were the culmination of an approximate
two year investigation by the Drug Enforcement Administration,
Muscogee County Sheriff's Office, and the Metro Task Force,
According to terms stipulated at sentencing, HILL participated in a
series of five cocaine sales to an informant which occurred between
March 3, 2005 and April 26, 2005. As a result of these sales, agents
obtained search warrants which led to the recovery of approximately
210 kilograms of cocaine and approximately 840 kilograms of
marijuana.
After HILL'S release on bond, he continued to direct the
distribution of cocaine and the collection of money resulting from
this distribution. Between September 2005 and February 2006, the
defendant was responsible for the distribution of at least 100
additional kilograms of cocaine and the collection of at least one
million dollars in illegal proceeds. At the time of his February 7,
2006 arrest, HILL possessed a ledger that indicated he was owed in
excess of two million dollars for distributed cocaine. During the
life of the investigation, seizures in the case included a total of
286 kilograms of cocaine, , 2,564 pounds of marijuana and over
$750,000.00 in U.S. Currency.
The prosecution was handled by Assistant United States Attorney Mel
Hyde. Any questions concerning this matter should be directed to Sue
McKinney, Public Affairs Specialist, United States Attorney's
Office, at (478)621-2602.
```

1

Figure 10.4: Example XML-file from the gold standard for Relation Extraction

```
<?xml version="1.0" encoding="UTF-8" ?>
<RE>
<TEXT><![CDATA[
For Immediate Release:
Thursday, January 24, 2008
Contact:
Douglas S. Collier
Public Information Officer
(973) 776-11
43
 (Office)
(862) 849-9
833
 (Cell)
DEA, HIDTA, Union County, Elizabeth, Hillside, Linden PD's Together Execute Operation: "Old School"
JAN 24  -- (Newark, NJ) - Gerard P. McAleer, Special Agent in Charge of the Drug Enforcement Administration (DEA) New Jersey Division, and Union County Prosecutor Theodore J. Romankow announced today the dismantling of three (3) distinct illegal
    narcotics enterprises in the City of Elizabeth, New Jersey. This was a seven month investigation which resulted in 18 individuals arrested facing both state and federal charges.
The investigation focused on the mid-level dealers who provide the link between the upper-level dealers who imported illegal narcotics into the country and the street-level dealers who peddle it on corners of crime-ridden neighborhoods.
The most recent arrests occurred this morning when at 5:45 a.m. over 100 police officers from 7 different law enforcement agencies, including 5 SWAT teams, conducted simultaneous raids on 8 locations in Elizabeth. Eleven people were arrested, 4 guns
    confiscated and heroin with an approximate street value of $116,000+, cocaine with a street value of $100,000 and $15,300 in cash were seized. The 400 grams of heroin seized was in a pure form and requires laboratory analysis to determine its
    degree of purity. Including the arrests and seizures that occurred in October, December and early January as part of the same investigation, the total amount of seizures rises to $203,500+ worth of heroin (street value), $100,000 worth of cocaine
    (street value), $30,500 in cash and 4 guns with a total of 19 people arrested.
According to Gerard P. McAleer, Special Agent in Charge of the DEA's New Jersey Division, "This investigation knows no boundaries. It extends beyond Elizabeth, beyond Union County and beyond New Jersey. We will continue to investigate this to the
    source country."
Anibal Rodriguez' street-level dealers were also identified and charged. Charged are his brother, Alniva "Julio" Rodriguez, Lamont Long, Rocky Lyles, Craig Pearson, Barry Warren and Dijuant Williams.
Two of Rodriguez' suppliers have also been identified as Curtis Neal of Perthy Amboy, and Ramon Reyes were arrested and charged in December when they purchased 150 bricks of heroin (7500 individual bags) from a New York supplier. That heroin had a
    street value of $75,000." The man from New York from whom he purchased the drugs has also been arrested and charged. Neal and Reyes are currently facing federal charges.
One of Rodriguez' street-level dealers is identified as Rocky Lyles. Lyles was arrested and charged in October, 2007 after he attempted to retrieve the jacket of a three-year-old girl from a day care center in Elizabeth. The jacket belonged to the
    daughter of Lyles' girlfriend. The child had already handed the 27 bags of heroin she found in her pocket to her teacher who alerted police. Among other things, Lyles is charged with endangering the welfare of a minor.
Karnell Wilson has also been identified as a supplier for Anibal Rodriguez. Wilson is part of an organization centered around Anthony Hopson. This organization provides heroin primarily to the area of Jackson Avenue and Bond Street and Jefferson Park
    in Elizabeth. Hopson is joined by his nephew, Khiah Hopson originally from Georgia who acts as his right-hand man. Wilson and Anthony and Khiah Hopson supplied and shared suppliers with other mid-level drug dealers inside and outside of Elizabeth.
    Some of their "customers" include Robert "Bobby" Franklin, Ryshane Graves and Manuel Veras---all of whom have been arrested and charged.
This investigation involved the cooperation of several agencies besides the others already mentioned. These include: Newark Police Department, the U.S. Marshal's Service and officers from various local police departments throughout the state who are
    assigned temporarily to the DEA's High Intensity Drug Trafficking Area Task Force.
Despite these charges every defendant is presumed innocent, unless and until found guilty beyond a reasonable doubt, following a trial at which the defendant has all of the trial rights guaranteed by the United States Constitution and State law.
]]></TEXT>
<TAGS>
<relation id="r0" start="840" end="874" text="street-level dealers who peddle it" type="sell" />
<arg1 id="a0" start="840" end="860" text="street-level dealers" />
<arg2 id="a0" start="872" end="874" text="it" />
</TAGS>
</RE>
```

Bibliography

[AG00] E. Agichtein, L. Gravano. “Snowball: Extracting relations from large plain-text collections.” In: *Proceedings of the fifth ACM conference on Digital libraries*. ACM. 2000, pp. 85–94 (cit. on p. 20).

[al13] T. M. et al. *word2vec*. Apache License 2.0. 2013. URL: https://code.google.com/archive/p/word2vec/ (cit. on pp. 42, 47, 62).

[BDVJ03] Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin. “A neural probabilistic language model.” In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155 (cit. on p. 17).

[BL+07] Y. Bengio, Y. LeCun, et al. “Scaling learning algorithms towards AI.” In: *Large-scale kernel machines* 34.5 (2007), pp. 1–41 (cit. on p. 17).

[BM57] F. Bacon, B. Montagu. *The Works of Francis Bacon*. Vol. 1. Parry & McMillan, 1857 (cit. on p. 9).

[DJHM13] J. Daiber, M. Jakob, C. Hokamp, P. N. Mendes. “Improving efficiency and accuracy in multilingual entity extraction.” In: *Proceedings of the 9th International Conference on Semantic Systems*. ACM. 2013, pp. 121–124 (cit. on p. 14).

[GE10] Y. Goldberg, M. Elhadad. “An efficient algorithm for easy-first non-directional dependency parsing.” In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics. 2010, pp. 742–750 (cit. on p. 29).

[GH14] R. Grishman, Y. He. “An information extraction customizer.” In: *International Conference on Text, Speech, and Dialogue*. Springer. 2014, pp. 3–10 (cit. on p. 23).

[GL14] Y. Goldberg, O. Levy. “word2vec Explained: deriving Mikolov et al.’s negative-sampling word-embedding method.” In: *arXiv preprint arXiv:1402.3722* (2014) (cit. on p. 19).

[Gol14] Y. Goldberg. *word2vecf*. Last updated 01/01/2018. 2014. URL: https://bitbucket.org/yoavgo/word2vecf (cit. on p. 63).

[Gri] R. Grishman. *Java Extraction Toolkit*. Apache Software Foundation License 2.0. URL: http://cs.nyu.edu/grishman/jet/jet.html (cit. on pp. 28, 60).

[Har54] Z. S. Harris. "Distributional structure." In: *Word* 10.2-3 (1954), pp. 146–162 (cit. on p. 29).

[HG15] Y. He, R. Grishman. "ICE: Rapid Information Extraction Customization for NLP Novices." In: *HLT-NAACL*. 2015, pp. 31–35 (cit. on p. 22).

[LG14] O. Levy, Y. Goldberg. "Dependency-Based Word Embeddings." In: *ACL (2)*. 2014, pp. 302–308 (cit. on p. 30).

[MBSJ09] M. Mintz, S. Bills, R. Snow, D. Jurafsky. "Distant supervision for relation extraction without labeled data." In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*. Association for Computational Linguistics. 2009, pp. 1003–1011 (cit. on p. 21).

[MCCD13] T. Mikolov, K. Chen, G. Corrado, J. Dean. "Efficient estimation of word representations in vector space." In: *arXiv preprint arXiv:1301.3781* (2013) (cit. on pp. 18, 30).

[MG11] B. Min, R. Grishman. "Fine-grained entity set refinement with user feedback." In: *Information Extraction and Knowledge Acquisition* (2011), p. 2 (cit. on pp. 14, 41).

[MGD16] O. Melamud, J. Goldberger, I. Dagan. "context2vec: Learning generic context embedding with bidirectional lstm." In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. 2016, pp. 51–61 (cit. on p. 16).

[MKB+10] T. Mikolov, M. Karafiát, L. Burget, J. Cernocky, S. Khudanpur. "Recurrent neural network based language model." In: *Interspeech*. Vol. 2. 2010, p. 3 (cit. on p. 17).

[MSB+14] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, D. McClosky. "The stanford corenlp natural language processing toolkit." In: *ACL (System Demonstrations)*. 2014, pp. 55–60 (cit. on pp. 61, 62).

[NWV17] K. A. Nguyen, S. S. i. Walde, N. T. Vu. "Distinguishing antonyms and synonyms in a pattern-based neural network." In: *arXiv preprint arXiv:1701.02962* (2017) (cit. on p. 21).

[PCB+09] P. Pantel, E. Crestan, A. Borkovsky, A.-M. Popescu, V. Vyas. "Web-scale distributional similarity and entity set expansion." In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*. Association for Computational Linguistics. 2009, pp. 938–947 (cit. on p. 20).

[PP06] P. Pantel, M. Pennacchiotti. “Espresso: Leveraging generic patterns for automatically harvesting semantic relations.” In: *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. Association for Computational Linguistics. 2006, pp. 113–120 (cit. on p. 15).

[RH02] D. Ravichandran, E. Hovy. “Learning surface text patterns for a question answering system.” In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 41–47 (cit. on pp. 14, 15).

[SG10] A. Sun, R. Grishman. “Semi-supervised semantic pattern discovery with guidance from unsupervised pattern clusters.” In: *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*. Association for Computational Linguistics. 2010, pp. 1194–1202 (cit. on pp. 20, 21).

[SGD16] V. Shwartz, Y. Goldberg, I. Dagan. “Improving hypernymy detection with an integrated path-based and distributional method.” In: *arXiv preprint arXiv:1603.06076* (2016) (cit. on pp. 22, 53, 66).

[Tea] D. D. Team. *Deeplearning4j: Open-source distributed deep learning for the JVM*. Apache Software Foundation License 2.0. URL: http://deeplearning4j.org (cit. on p. 29).

All links were last followed on July 19, 2017.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature