

# Grounded Learning for Source Code Component Retrieval

Ákos Kádár

THESIS SUBMITTED IN PARTIAL FULFILLMENT  
OF THE REQUIREMENTS FOR THE DEGREE OF  
MASTER OF SCIENCES IN COMMUNICATION AND INFORMATION SCIENCES,  
MASTER TRACK HUMAN ASPECTS OF INFORMATION TECHNOLOGY,  
FACULTY OF HUMANITIES  
TILBURG UNIVERSITY

Thesis committee:

Grzegorz Chrupala

Afra Alisahi

Tilburg University

Faculty of Humanities

Department of Communication and Information Sciences

Tilburg, The Netherlands

August 2014

## Acknowledgment

I would like to thank my supervisor Grzegorz Chrupala for giving me freedom to explore the topic and bringing me to TiCC to present my thesis, which gave me a great opportunity to get valuable feedback. I would also like to thank him for providing an exciting topic, a good data set, numerous references and helping with the implementation. Furthermore, I am grateful for his attention to detail and unforgiving commitment to sound methodology. Writing the thesis under Grzegorz's supervision and working for him as a research assistant had a great impact on my attitude towards science. I would also like to thank Leif Johnson for his personal support with the theano-nets module and for implementing my feature requests. Zoltán Varjú helped a lot during this year and I cannot thank him enough for his inexhaustible support and good friendship. He was the one who recommended Tilburg University to me and taught me a lot about NLP and made me excited about the field. I would also like to thank my parents for the financial support, my mum for providing emotional support during the harder phases of my project and my dad for making me excited about moving to other countries to learn and for his active role in planning my education and my life in general. I owe a great deal to the Nyelvtudományi Intézet and in particular to the Elméleti Nyelvészet Tanszék for teaching me almost everything I know and care about and for surviving the conditions they've been facing for a long time. I would like to thank the extended family of 397, Gabriele, Talin, Dene, Bilal and Mirsim who made the grey, rainy days in Tilburg much brighter and shiner with good food, couple of beers and interesting conversations. They say "True friendships continue to grow even over longest distances". We will keep and feed our family-friendship until our next reunion. Finally, thank You Piszi for being the greatest person I've ever known and for being intelligent, caring, tolerant and honest and making life worth living!

Á.K.

# Contents

Acknowledgment . . . . .	i
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>4</b>
2.1 Outline . . . . .	4
2.2 Context-theoretic Semantics . . . . .	4
2.3 Computational Distributional Semantics . . . . .	5
2.3.1 Overview of the vector-space model . . . . .	5
2.3.2 Meaning captured in Hyperspaces . . . . .	7
2.3.2.1 Hyperspace Analogue to Language . . . . .	8
2.3.2.2 Latent Semantic Analysis . . . . .	8
2.3.3 Latent Dirichlet Allocation . . . . .	10
2.4 Grounding . . . . .	11
<b>3 Previous Work</b>	<b>13</b>
3.1 Grounding by Translation . . . . .	13
3.2 Source-Code Retrieval . . . . .	15
3.2.1 LSI in Source-Code Retrieval . . . . .	17
3.2.2 LDA in Source-Code Retrieval . . . . .	18
3.3 Source-Code Retrieval as Translation . . . . .	20
3.3.1 Previous Work . . . . .	21
<b>4 Methods and Data set</b>	<b>23</b>
4.1 Outline . . . . .	23

4.2	Data	23
4.3	Feature vectors	24
4.4	Translation Models	26
4.4.1	Linear models	27
4.4.2	Multilayer Perceptron	28
4.4.3	Training the neural network translation model	28
4.5	Ranking	32
<b>5</b>	<b>Experiments</b>	<b>33</b>
5.1	The setup	33
5.1.1	Data sets	33
5.1.2	Queries	33
5.2	Evaluation	34
5.3	Vector Representation	34
5.4	Retrieval engine	36
5.5	Parameter tuning for training	36
5.5.1	Training the Linear Models	36
5.5.2	Neural Networks Training	36
5.5.2.1	Implementations	36
5.5.2.2	Network Architecture	38
5.5.2.3	Hyper-parameter tuning	39
<b>6</b>	<b>Results</b>	<b>42</b>
6.1	Results on Retrieval	42
6.2	Word Similarity	43
6.3	Prototype	44
<b>7</b>	<b>Future work and Conclusion</b>	<b>47</b>
7.1	Future work	47
7.1.1	More realistic evaluation	47
7.1.2	Extensive tuning of the Multilayer Perceptron	48
7.1.3	Web-interface	48

7.2 Conclusion ..... 49

**References** ..... **50**

# List of Figures

2.1	Joint probability distribution in LDA . . . . .	10
4.1	Original Description-Signature pairs . . . . .	25
4.2	Description-Signature pairs after preprocessing . . . . .	26
4.3	Multilayer Perceptron with one Hidden Layer . . . . .	30
4.4	Model Creation . . . . .	31
5.1	Impact of $\alpha$ on <i>MRR</i> on the validation set . . . . .	37
5.2	Effect of the number of neurons on execution time on the validation set . . . . .	39
5.3	Effect of the learning-rate on <i>MRR</i> on the validation set . . . . .	40
5.4	Effect of the learning-rate . . . . .	41
6.1	Searchbox with link to the repo . . . . .	45
6.2	Dialogue box with the suggested query . . . . .	45
6.3	Search results . . . . .	46

# List of Tables

- 2.1 Tf\*idf Matrix . . . . . 7
  
- 5.1 The effect of vector representations on the *MRR* on the validation set . . . . . 35
- 5.2 Effect of the number of hidden-neurons on *MRR* . . . . . 39
  
- 6.1 Comparison of the models on the validation set . . . . . 43
- 6.2 Word Similarity in the hidden layer . . . . . 44
- 6.3 Time related terms . . . . . 44

## Abstract

In my thesis project I show that by modelling Source Code Retrieval as a regression problem: a) comparable results can be achieved to previous work by (Deng & Chrupala, 2014) b) The meaning of English words can be grounded in Java method-signatures. I evaluate these regression techniques on both retrieval and grounding tasks. In contrast to traditional static source code retrieval - where pieces of source code from a particular application are retrieved - I focus on retrieving source code components: meaningful elements of a programming language. The presented models are able to retrieve Java method-signatures given English queries. The documentation of the Java Standard Library is treated as a parallel corpus between Java method-signatures and English as the Javadoc format includes method name, qualifier, return type, parameter type, names for each method with the addition of a description. These descriptions and method-signatures are then represented in separate vector-spaces using various vector representations and a regression model is trained to predict a method-signature-vector given a description-vector. To use the regression model for retrieval, natural language queries are converted to description-vectors and are used as an input to the regression models which predict a signature-vector. Given the predicted signature vector all other signatures from the collection are ranked according to their cosine similarity to the query. Regression models used as translation models include: Linear Regression, Ridge Regression and Multilayer Perceptron. Models were evaluated on Mean Reciprocal Rank, Accuracy at 1 and Accuracy at 10. The best performing model was the Ridge Regression with complexity parameter  $\alpha = 0.2$ ,  $MRR = 0.389$ ,  $Acc@1 = 0.232$  and  $Acc@10 = 0.707$ . To test how well the models perform on grounding a matrix constructed from all the English word-vectors is translated using the regression models. After translation lists of similar words are constructed using pairwise cosine-similarity. This leaves us with word similarity lists and qualitative analysis of the word similarity results is provided.



# 1 | Introduction

Language traditionally is described on various linguistic levels. Phonology and phonetics explore the regularities and properties of the sounds of linguistic expressions, morphology analyzes larger meaningful units such as words and syntax describes the structure of multi-word constructions such as sentences. Semantics on the other hand deals with *meaning*. Meaning is one of the oldest and most daunting problems of linguistics and there have been countless proposals to describe how the meaning of linguistic expressions are a) related to the external world, b) related to inner mental states and c) related to each other. Distributional Semantics is a framework that is concerned with problem c and its main aim is to automatically extract relations between words from large corpora. The underlying intuition of this approach is best described by the famous quote from Firth (1957): "You shall know a word by the company it keeps". Distributional Semantics tackles the problem of meaning in a data-driven computational fashion:

1. It takes large quantities of texts as input
2. Models the semantics using mathematical models such as linear-algebra and probability theory
3. Implements these models in software systems
4. Analyzes the data computationally
5. Outputs a representation modelling the relationships between words or larger units

This approach to tackle semantics proved to be of unprecedented power in that it provided a useful representation for both Natural Language Processing applications and for Cognitive Linguistics research, while relying on very general and widely applicable and flexible formalisms.

However, a major limitation of traditional Distributional Semantics is that it only deals with the relationships between linguistic expression and does not relate them to any sort of external reality. The fundamental question that is explored in the present thesis is how to exploit the methods of Distributional Semantics to ground the meaning of linguistic expressions in extra-linguistic reality. Traditionally semantics deals with the issue of grounding by representing natural language utterances with logical forms and mapping these logical forms to mathematical models such as set theory or category theory. The theoretical vision behind the thesis is to swap-out logical forms with a programming language which already has an underlying model based on computational constructs. As the underlying model for programming languages is given the grounding problem is reduced to finding a mapping between natural languages and programming languages. Taking inspiration from Distributional Semantics this problem can be tackled by exploiting the co-occurrences between natural language words and programming language terms. The aim of the work of [Deng and Chrupała \(2014\)](#) is to explore the possibility of grounding the meaning of natural language expressions in programming language components as well as to create an Information Retrieval model for retrieving programming language components. Again, this is in line with the traditional Distributional Semantics idea in that the goal is to provide a meaning representation which is both theoretically plausible and practically useful. More precisely the system in [Deng and Chrupała \(2014\)](#) is able to retrieve Java methods from the Java Standard Library given English queries and to associate English expressions with Java methods. I base my work on [Deng and Chrupała \(2014\)](#) and provide a novel approach to handle the same problem. Furthermore, I test how well the model captures the meaning of English expression by performing word-similarity experiments. During the project I made the following contributions:

- Regression models were applied to Source Code Component retrieval.
- The baseline in [Deng and Chrupała \(2014\)](#) was met and one of the two models were outperformed.
- The models' performance on both retrieval and grounding tasks were evaluated.
- A simple graphical user-interface was created for the retrieval engine.

The goal is to provide a system where developers can query in natural language and receive appropriately ranked methods in the results, but also to provide an effective framework to ground the meaning of linguistic expressions. More and more problems that humans face are expressed in both natural language and programming language e.g. the documentation or requirements of an application describes the formulation of the problem and the programming language describes a candidate solution. Taking these texts as a natural-formal language parallel corpora would provide a great opportunity to ground the meaning of natural language in some language-external reality using proven and effective methods.

## 2 | Background

### 2.1 Outline

This chapter introduces the reader to the theoretical background of the project. Section 2.2 introduces the context-theoretical approach to meaning - one of the main theoretical backbones of the project - while Section 2.3 describes its various implementations. These sections relate their main contents to Information Retrieval as one of the products of my Thesis project is a Source Code Component Retrieval model for Java methods. Section 2.4 introduces the other important theoretical foundation of the Thesis namely the problem of grounding linguistic expressions in extra-linguistic reality.

### 2.2 Context-theoretic Semantics

The terms Distributional-, Corpus-based-, Context-theoretic-, or Statistics-based semantics refer to the approaches that share a usage-based or empirical view of semantics research and meaning in general. The underlying assumption is that the distribution and general statistical properties of words - or groups of words - play an important role in their meaning. This notion is based on the Distributional Hypothesis (DH): “The degree of semantic similarity between two linguistic expressions A and B is a function of the similarity of the linguistic contexts in which A and B can appear.” (Lenci, 2008). In principle it follows from the DH that if it is true that the context does encode certain aspects of word meaning, and that meaning is actually dependent on the context, than the larger, more representative corpus we have, the more information we can potentially uncover about word meaning.

The DH might seem to be a modern approach of computational linguistics, but distributional analysis as a methodology for linguistics was proposed by Harris in the earlier days of American structuralism (Nevin, 2002). His distributional methodology was first applied to phonemic analysis and later has been generalized to multiple levels of linguistic analysis. It was Harris' theory that the degree of similarity between words is the function of the similarity of the context they appear in Harris (1954). He also claimed that distribution should be taken as explanation for meaning and that similarity classes can be constructed based on co-occurrence statistics. One of the most typical aspects of context-theoretic semantics is that distributional models are driven by empirical observations. This data-driven approach to linguistic theory is what this thesis adopts in its methodology. A shortcoming of distributional approaches, however, is that they do not link linguistic expressions to extra-linguistic objects.

Indeed most of the work in related fields such as vector space semantics, only take into account the frequencies of words in some kind of texts e.g. documents, reviews. This is largely due to the fact that the technology for extracting features from texts is far more advanced than the technology for extracting features from images or videos (Lenci, 2008). Recently, however, there has been research in using techniques from text based distributional semantics for extracting word co-occurrences with image features (Mathe, Fazly, Dickinson, & Stevenson, 2008). In the present thesis this approach to co-occurrence based semantics is adopted and English descriptions of methods provide the linguistic expressions while the co-occurring Java methods are taken as extra-linguistic objects. A closely related approach to the present thesis involves grounding natural language in formal languages and is generally referred to as Semantic Parsing. Some of the most relevant work in this field is described in Section 3.1. The work that I closely follow in my thesis and which is the most relevant is by Deng and Chrupała (2014).

## 2.3 Computational Distributional Semantics

### 2.3.1 Overview of the vector-space model

A family of techniques in Natural Language Processing which rely on the distributional hypothesis are often called vector semantics or vector-space semantics. This section concentrates on

the description of these techniques in the context of Information Retrieval. The type of IR techniques that are relevant to the thesis aim at retrieving information given some information need from a corpus of unstructured data. To be able to find relevant information first the sources of information must be gathered and indexed. One of the most important concepts in IR is similarity and in the vector-space model (VSM) of IR the similarity between the query and the document is measured and the documents are ranked according to their similarity to the query. The document collections needs to be indexed and represented in a way that it enables the system to compute similarity between queries and documents. Arguably the most popular way is to store the  $tf \cdot idf$  score for every term  $t$  in the high dimensional term-frequency matrix.  $Tf \cdot idf$  stands for term-frequency multiplied by inverse document frequency. Term frequency is the number of times the term  $t$  occurs in the document  $D$ , but usually instead of the raw score the  $\log(1 + tf)$  is stored - it can be also normalized by the number of words in the document as follows:

$$\frac{\log(tf)}{N_{term}}$$

Inverse document frequency is the total number of documents divided by the number of documents the term occurs in.

$$\log\left(\frac{N_{doc}}{1 + |d \in D : t \in d|}\right)$$

Table 2.1 shows the  $tf \cdot idf$  weighted term-document matrix for a document collection consisting of two documents  $d_1 = \{i, am, a, document, and, i, am, short\}$  and  $d_2 = \{being, short, is, not, a, bad, thing\}$ . Based on the  $tf \cdot idf$  vectors the similarity between query  $q$  and document  $d$  is computed as the cosine between two vectors (Singhal, 2001) calculated as:

$$\cos(\theta) = \frac{d \cdot q}{\|d\| \|q\|}$$

$d \cdot q$  is the dot product of the document-vector and the query-vector and  $\|q\|$  is the norm of the query vector.

$$\|q\| = \sqrt{\sum_{i=1}^n q_i^2}$$

The notion of term and similarity are not inherent in the vector-space model (Singhal, 2001). Terms can be stemmed or tokenized words, phrases or any other features. Furthermore, instead

Table 2.1: Tf\*idf Matrix

	$D_1$	$D_2$
I	0.227	0
am	0.227	0
a	0.09	0.09
document	0.143	0
and	0.143	0
short	0.09	0.09
being	0	0.143
is	0	0.143
not	0	0.143
bad	0	0.143
thing	0	0.143

of using the cosine similarity different similarity measures could be used as well e.g. euclidean distance. The described vector-space model of Information Retrieval relies on term matching and assumes that words are semantically independent and therefore do not provide a model of meaning. Semantic relatedness is out of the picture in this framework, which is contra-intuitive since we know that documents with similar terms contain similar information. The following sections introduce models which extract latent semantic features from the described vector-spaces.

### 2.3.2 Meaning captured in Hyperspaces

Although, in relation to IR vector-space semantics models described in Section 2.3.2.1, 2.3.2.2 and 2.3.3 are usually treated as useful tools to handle synonymy and polysemy and in general to improve search performance by introducing the notion of semantic relatedness, they form a family of full-fledged approaches to study the meaning of linguistic expressions. These high-dimensional theories of meaning model the meaning of the elements of a language as relations between abstract, amodal and arbitrary symbols (Glenberg & Kaschak, 2002), implying that the meaning of utterances is a function of the meaning of the elements and some sort of syntactic

combination of these arbitrary symbols. Symbols are abstract in the sense that the term "table" is used for a large variety of referents, amodal in that "table" means the same written or spoken and arbitrary in that the form of "table" bears no relationship to its meaning. Section 2.3.2.1 and Section 2.3.2.2 describes two major vector-space semantics models. In both models the meaning of a term is defined by its relation to all other terms in the constructed matrices.

### 2.3.2.1 Hyperspace Analogue to Language

In the HAL framework a word co-occurrence matrix is extracted from a large collection of texts using a sliding window of some  $n$  words (Lund & Kevin, 1997). Each row and column in the  $n \times n$  matrix is labeled by a word and the cells contain information about how close these words are together - words adjacent to each other receive a score equal to the size of the sliding window  $n$ , words with the distance of 1 receive  $n - 1$  and so on. The rows of the matrix give the co-occurrence values for terms preceding the row-labels and columns give the co-occurrence values for terms following the column-labels. As mentioned in the tf\*idf example, the similarity between terms can be computed by the cosine-similarity measure. Under this framework words are similar because they share similar contexts, which makes HAL a straightforward implementation of context-theoretic semantics. It was also a great breakthrough as it offers a completely data-driven lexical semantic theory and excludes any bias from humans provided that the corpus is large enough and unbiased. Lund, Burgess, and Atchley (1995) also demonstrated that the rate of similarity among word vectors correlated with degree of priming in a lexical decision task.

### 2.3.2.2 Latent Semantic Analysis

Latent Semantic Analysis or Latent Semantic Indexing is also based on the fundamental notion of DH that words with similar contexts have similar meanings and is widely used in IR literature. The fundamental notion of LSI is that documents with numerous common words are conceptually similar. Rather than constructing word co-occurrence vectors as in the HAL framework, under this approach, a collection of documents is taken as an observation from which we should infer which terms belong to the same concepts, which terms are conceptually similar. The input for the LSI system is the term-document matrix augmented by some global weighting factor

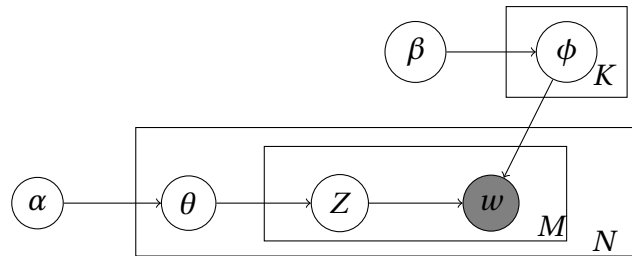


such as the tf\*idf weighting<sup>1</sup> (Berry & Browne, 1999). After creating the matrix, column vectors contain the information about the words in the documents and row vectors indicate which word occurs in which document. Since rows and columns represent relations between terms and documents, the term-document matrix is also a term co-occurrence matrix. When LSI is employed the co-occurrence matrix is mapped to a space with a smaller number dimensions - while trying to keep as much information as possible - using a linear dimensionality reduction technique called Singular Value Decomposition to discover associations. The resulting matrix is less sparse and the original weighted term counts are replaced by semantic similarity scores - representing similarities between terms - which encode the implicit latent structure of the associations between terms and documents. These similarity values can be positive or negative: the negativity means semantic distance while positivity means semantic similarity. In essence the LSI matrix encodes a concept of documents where the similarity between documents is based on their distance in the vector-space which represents their semantic/conceptual distance. When LSI is used for retrieving documents given a query the query is transformed into a vector in the low dimensional concept space and the documents are ranked based on their similarity to the query (Deerwester, Dumais, Landauer, Furnas, & Harshman, 1990). The fundamental advantage of using LSI over the regular tf\*idf counts is that it allows to extract concepts and semantic structure from an unstructured corpus of texts and in principle an LSI based system can return not only documents which contain the words of the query, but documents which are conceptually similar to the query (Dumais, Furnas, Landauer, Deerwester, & Harshman, 1988). As a theory of meaning it is similar to HAL in essence in that it also claims that the matrices created by LSI encode the meanings of the terms in a particular language: the actual meaning of a term is represented in an  $n$  dimensional space. In their article Landauer and Dumais (1997) the authors make claims such as LSA is "a possible theory about all human knowledge acquisition, as a homologue of an important underlying mechanism of human cognition in general" and also "a unified representation of knowledge". In their study they also showed that LSA can come close to perform in certain tasks just as a non-native English speaker e.g. in choosing synonyms for a variety of terms and they also showed that averaging sentence vectors can predict coherence judgements (Landauer & Dumais, 1997).

---

<sup>1</sup>As explained in section 2.3

Figure 2.1: Joint probability distribution in LDA



### 2.3.3 Latent Dirichlet Allocation

LSA and HAL have in common that they both rely on linear-algebraic representations of words and their contexts; hence the term vector-space semantics. The model described in this section is also a framework for context-theoretic semantics, however, rather than relying on linear-algebra it is a probabilistic model of word-meaning. More specifically it is a probabilistic generative model meaning that the data is treated as if it was generated by a probabilistic process. The goal is to infer the probability distribution that the data was generated by and uncover latent variables that correspond to the thematic structure of the document collection (Blei, Ng, & Jordan, 2003). In other words we need to compute the conditional probabilities of the hidden variables given the observation which is the document collection in our case. The assumption is that every document is comprised of multiple topics. Each topic is a distribution over all terms in the vocabulary. Different topics have different words with different probabilities. Documents are generated by choosing a distribution over topics and then picking words according to their probabilities given the topics. Similarly to HAL and LSI, LDA is a bag-of-words model meaning that the order of words is not taken into account. The goal of the inference procedure in LDA is to infer the underlying topic structure, their distribution over terms, and what topics individual documents are generated by. The  $K$  number of topics  $\phi_k$  of the whole document collection are drawn from a Dirichlet distribution with parameters  $\beta$  and are independent from all other random variables of the model. Similarly for each document from a number of  $D$  documents a set of topic proportions  $\theta_d$  is drawn from a Dirichlet distribution with parameters  $\alpha$ . Each document consists of  $N$  words and it is assigned a topic by the topic assignment  $Z_{d,n}$  drawn from  $\theta$  and so each word  $w_d$  in the document is dependent on the topic assignment  $Z_{d,n}$  as well as  $\beta_{1,k}$ . The plate diagram on Figure 2.1 provides a compact representation of the model.

As LDA is a highly flexible hierarchical Bayesian model it has been applied for a wide variety of tasks and has been further developed including several task-specific versions. Although, it was originally intended as a topic modelling framework it also became popular in other computational semantics applications. The word-topic distributions inferred by LDA are proposed by [Cai, Lee, and Teh \(2007\)](#) to be used as features for training a supervised word-sense disambiguation models. LDA has been employed by [Materna \(2012\)](#) to induce semantic frames (LDA-frames) which capture semantic information of triplets of subject-verb-object.

## 2.4 Grounding

The meanings of words are largely a function of their contexts<sup>2</sup>. Computational implementations exploiting the Distributional Hypothesis offer large improvements for information search problems and provide evidence and insight to cognitive linguistics research<sup>3</sup>. I would like to argue, however, that defining the relationships between symbols without mapping those to a model or "the reality" is limited in its utility. According to [Roy \(2005\)](#) creating models and systems that take language as a symbol manipulations system and do not connect these symbols to extra-linguistic reality suffer from limitations and that any theory of meaning that does not provide a non-linguistic foundation to meaning is "fundamentally limited". Symbols are primitive and undefined, only by the rules and by the way rules combine them they are systematically interpretable as having meaning. An arithmetic system is a good example of a symbol system. Numerals ("1", "2", "3") might be part of such a system and formal rules define ways to combine them into legal formulas. Making true statements based on the symbols and formal rules such as " $1 + 2 = 3$ " render the symbols systematically interpretable. Although, it might seem that the symbols outside of the symbol system actually have meaning, but that is only the case for the human reader as she is used to their everyday interpretation. For a calculator the symbols only have meaning within the system ([Harnad, 2003](#)). Natural language under the Distributional Semantics approach is such a symbol system with rules manipulating those symbols, therefore words do not refer to any entity in the "outside world". They are only interpretable within the natural language itself. For a symbol system to be grounded "It would have to be able to pick

---

<sup>2</sup>Section 2.2

<sup>3</sup>Sections 2.3.2.1, 2.3.2.2 and 2.3.3

out the referents of its symbols, and its sensorimotor interactions with the world would have to fit coherently with the symbols." (Harnad, 1990)

Trivially for any sort of artificially intelligent agent - e.g. robots - that interacts with humans in the physical - or some virtual - world, having knowledge about the language internal relationships of linguistic elements is not enough. They must be able to map linguistic expressions to the world they exist in e.g. consider the task of following directions or executing commands related to an office environment.

## 3 | Previous Work

### 3.1 Grounding by Translation

The most closely related work to the present thesis comes from Semantic Parsing. It concentrates on learning a mapping from natural languages to some sort of formal representations using statistical methods and innovative supervision techniques<sup>1</sup> (Richardson & Kuhn, 2014). The ultimate goal is to translate natural language to an unambiguous executable formal language that can be interpreted by a program to perform certain tasks. A typical direction taken by researchers in the field is to use some form of weak supervision where the formal meaning representation is an already existing domain specific formal language grounded in the concepts of the particular domain it describes and it is somehow already aligned with natural language expressions. Typical practical uses of such systems are natural language interfaces to structured databases and command languages (Mooney, 2007). Given the target application domain of such systems they are typically evaluated extrinsically.

There are several different corpora developed that were constructed from parallel natural language sentences and formal language expressions in domains such as geography and navigation instructions (Mooney, 2007). A couple of general frameworks for Semantic Parsing have been developed to learn statistical semantic parsers that are robust against noise and can potentially scale up to large data sets. Probably the most notable one is WASP, which is based on statistical-machine translation methods (Wong & Mooney, 2006). WASP does not need any knowledge about the natural language it only assumes that the target formal language is unambiguous and it requires a CFG that describes it. It builds a bilingual lexicon using GIZA++ to align natural

---

<sup>1</sup>Such as ambiguous supervision in Mooney (2008)

language words to formal language productions, which is then used to formulate production rules within a synchronous context-free grammar (SCFG) framework. These production rules re-write non-terminals to two strings - natural language and formal language - and therefore simultaneously produce both natural language sentences and their formal counterparts. The parameters for the probabilistic parser are learned by training a maximum-entropy model by expectation maximization. If the model is fully learned, given a natural language utterance, its most probable parse is found and its formal counterpart is returned. It has been applied by [Matuszek, Fox, and Koscher \(2010\)](#) to develop a system that lets robots learn from parallel corpora of formal and English path descriptions to follow directions given natural language commands.

In more recent work [Andreas, Vlachos, and Clark \(2013\)](#) experiment with using off-the-shelf statistical-machine-translation tools to perform semantic parsing on the GeoQuery data set. It consists of 880 pairs of natural-language questions (English, German, Greek and Thai), and their representations in a GeoQuery's own meaning representation language, which can be used to query a Prolog database interface. The main idea is similar to WASP as it is based on the observation that Semantic Parsing is essentially a translation problem from natural language utterances to statements in some formal language. First, formulas from the GeoQuery meaning representation language are converted to a more natural-language-like form: they take the pre-order traversal of the formulas and annotate the functions with the number of arguments they take, eliminating the need for bracketing, but keeping the unambiguous nature of the language, which allows to always reconstruct a tree from a given sequence. Similarly to WASP the next phase is to learn the alignments using IBM Model 4. They learn many-to-many alignments in both directions. For the translation rule induction they use both phrase-based and SCFG approaches. As for the learning phase they do not assume any knowledge about the syntax of the formal language, they learn an n-gram language model for the formal language from the training data. When predicting a sentence in the formal language they use both the translation and language model scores and filter the best candidate translations until a well-formed sequence is found. The last step is to convert the predicted formal sequence back to the original GeoQuery representation. Their results on the GeoQuery data set are comparable to other purpose built semantic-parsers and their system trains considerably faster.

So far there are only a few existing data sets for Grounded Learning ([Richardson & Kuhn, 2014](#)). Some of the data sets are too restricted to learn interesting general grounded meaning representations: the GeoQuery corpus mentioned above only contains 38 predicate-types and the Sportcaster ([Chen & Mooney, 2008](#)) corpus only has 9 types of relations and a few dozen types of entities. More recently there have been several attempts to create more large scale data corpora including the latest UnixMan corpus from [Richardson and Kuhn \(2014\)](#). This data set is very closely related to the present Thesis in that it is a resource for semantic parsing constructed from Unix man pages by taking English descriptions and command examples as a parallel corpus. Similarly, in my project the data set consists of Java method-signatures and their descriptions.<sup>2</sup>

## 3.2 Source-Code Retrieval

Searching for information, concepts or problems in documentations and source-code are some of the main activities of anyone who is involved in writing software or program code in general. Source-code retrieval is concerned with retrieving code fragments which are functionally or conceptually similar to each other or to a query. This can be done statically and dynamically. Whereas in case of dynamical code retrieval the information content of the program code is assessed during runtime, static analysis concentrates on directly analyzing the source-code itself. My discussion about source-code retrieval concentrates on static techniques as it is more closely related to my thesis project. The most traditional techniques used in static source-code retrieval include regular expression matching tools like Unix grep ([Poshyvanyk & Marcus, 2007](#)). However, to enable more elaborate search techniques semantic information can be extracted from the source-code in form of identifiers, comments, string literals or pieces of the code itself. The fundamental idea is that the knowledge of the developers is encoded in the comments, identifier names etc. and better software analysis can be achieved by mining this knowledge. For different source-code retrieval applications the code can be parsed on different levels of granularity - classes, methods - depending on the users' needs. After parsing the source-code it is

---

<sup>2</sup>Sections [4.2](#) and [5.1.1](#) explain the data set in detail.

broken down into source-code documents, where the code snippets are the documents and the code-internal documentation in natural language serves as the features for these documents. Various pre-processing steps are used on the natural language sentences before representing the resulting document collections using tf\*idf matrices<sup>3</sup>. These pre-processing steps include stemming, lower-casing, stop-word-filtering and splitting up identifiers and other expressions on hyphens, underscores or use camel-case-splitting e.g "FooBar" or "FOO\_bar" or "foo-Bar" to "foo", "bar". Both the tf\*idf representation and pre-processing techniques are very typical in IR applications. However, some source-code retrieval specific techniques can also be involved in the indexing process such as classes inheriting a portion of the vocabulary of their superclasses as in [Kuhn, Ducasse, and Girba \(2007\)](#). After the pre-processing and vectorization steps recent research in source-code retrieval has focused on IR models which are based on co-occurrence statistics and have the expressive power to represent the notion of meaning in terms of distributional semantics and concepts. Distributional Semantics based IR techniques such as LSI<sup>4</sup> and LDA<sup>5</sup> have been shown to cause major improvements in performance in many fields of source-code retrieval.

One of these fields is bug-localization, which is concerned with creating technology for automating maintenance tasks in software development by directly identifying the starting point in the source-code from which the code needs to be corrected. In this area a number of techniques have been explored e.g. dependency graphs, regular expressions, but models using IR techniques such Latent Semantic Indexing (LSI) as in [Liu, Yan, Fei, Han, and Midkiff \(2005\)](#) and Latent Dirichlet Allocation (LDA) as in [Lukins, Kraft, and Etzkorn \(2008\)](#) proved to yield better results and also allow users to query with the descriptions of the bugs in natural language. Section [3.2.2](#) demonstrates a case study where the researchers used LDA for bug-localization. Concept location, feature location or concept assignment<sup>6</sup> is another area that benefits from such IR techniques and it is concerned with retrieving parts of a source-code which implements a specific concept ([Marcus, Rajlich, Buchta, Petrenko, & Sergeev, 2005](#)). This can be

---

<sup>3</sup>The tf\*idf weighting scheme and the process of indexing is explained in Section [2.3](#)

<sup>4</sup>Described in detail in Section [2.3.2.2](#)

<sup>5</sup>Described in detail in Section [2.3.3](#)

<sup>6</sup>In some literature these terms refer to same field in other papers they might refer to slightly different tasks. For a comprehensive survey I refer the reader to [Dit, Revelle, Gethers, and Poshyvanyk \(2013\)](#).



useful to identify units of the source-code that implement certain feature of a given application or to facilitate code re-use over multiple projects. Some of the work in the area on top of code-internal documentation use other artifacts such as bug-reports and mailing-lists (Cleary & Exton, 2006). By using both code-internal and code-external artifacts Cleary and Exton (2006) developed the Eclipse Cognitive Assignment Plug-in, which is one of the state-of-the-art frameworks for concept location. In their work Gay, Haiduc, Marcus, and Menzies (2009) implement relevance feedback to the concept location system. In this framework after the ranked results are returned the user has the opportunity to rate the returned documents as being relevant or irrelevant. The system reformulates the query based on the feedback, returns the new results and the user can give feedback again. Related to concept location is the identification of high-level concept clones, where the aim is to identify code duplicates which are not a copy of a given piece of code, but a re-phrasing of it. A case study for the identification of high-level code clones is presented in Section 3.2.1.

### 3.2.1 LSI in Source-Code Retrieval

LSI<sup>7</sup> was successfully utilized in areas of source-code retrieval where identifying concepts and similarities between larger structures is essential e.g. it proved to give good accuracy when identifying duplicates of pieces of source-code in development projects as in Marcus and Maletic (2001). In their research they aimed at identifying duplicates of source-code which are not near exact copies, but are structurally different and yet solve the same or almost the same problems. To make this possible they extracted semantic features from the source-code and then broke it down into source-code documents. In their approach source-code documents are contiguous sets of source-code lines such as functions, blocks of declarations or class declarations represented by feature vectors extracted from their internal documentation e.g. from identifiers, string literals and comments. They ran LSI on these extracted semantic features and created an LSI representation of the document collection to encode the semantic similarity between the different source-code documents, therefore between the different units of source-code. Incorporating the LSI results they represented the software system as a relationship graph where

---

<sup>7</sup>Description of LSI is found in Section 2.3.2.2

the nodes are the source-code documents and weighted edges represent relationships between them. In the graph edges represent semantic and structural similarity and source-code documents were then clustered together based on their combined semantic-structural similarity score within the software system<sup>8</sup>. After the source-code documents were clustered the user could query with a file or a group of source-code documents to find the related high-level clones. The system was tested on Mosaic 2.7 an application written in C consisting of 95,000 lines in 269 files. They reduced the dimensionality of the original vocabulary of 5,114 terms to 350 using LSI for the 2,347 source-code documents and they managed to find 11 high-level concept clones implementing linked lists. In comparison with their term-matching - they have searched for keywords such as 'list' - and regular expression matching baseline the technique based on LSI reduced the search by a factor of five. Using LSI has also shown to be effective for bug localization in [Marcus et al. \(2005\)](#); [Marcus, Sergeev, Rajlich, and Maletic \(2004\)](#). They point out that a significant advantage that LSI can bring to the field is that an LSI-based source-code search engine can allow developers to use their regular search habits they got used to by using regular search engines like Google or Youtube.

### 3.2.2 LDA in Source-Code Retrieval

In source-code retrieval literature Latent Semantic Indexing is one of the most popular techniques borrowed from regular information retrieval, but more recent research showed that with Latent Dirichlet Allocation(LDA) better results can be obtained. Both LSI and LDA are suited for representing synonymy, but polysemy is better expressed with LDA<sup>9</sup>. In addition the matrix representation makes LSI a non-transparent model i.e. numeric spatial representation is used to model documents, queries and the results while LDA allows for the analysis of the document collection in terms of more human readable semantics as it describes topics as probability distributions over words ([Blei et al., 2003](#)). LDA was successfully utilized by [Lukins et al. \(2008\)](#) to localize bugs and they showed that LDA significantly outperforms LSI in bug-localization tasks. They extracted semantic information from the source-code including comments, identifiers and string literals on the granularity level of methods and performed stemming and stop-

---

<sup>8</sup>The measure used for structural similarity is not described here as it is not related to IR techniques. For more information please refer to the original article ([Marcus & Maletic, 2001](#))

<sup>9</sup>For more information on LDA please consult Section [2.3.3](#)

word elimination. This way in their document collections each document represents a method from the source-code and each document consists of terms from the comments, identifiers and string literals. This document collection was the input to the LDA tool GibbsLDA++ and the output of the process was a LDA model consisting of a word-topic probability distribution and a topic-document probability distribution. The queries in their case studies consist of terms describing the bug in English and the methods which need bugfixes should be returned given the query. The system was tested on 1.5 release 5 (1.5R5) of the software system Rhino focusing on bugs on method-level which bugs were fixed in either version 1.5R5.1 or v1.6R1 and only valid bugs that had been fixed were returned. Using this approach the system of [Lukins et al. \(2008\)](#) achieved good results: the relevant method was retrieved in top ten in case of 77% of the bugs, top five for 63% of the bugs, top result for 23% of the bugs.

The work from [Maskeri, Sarkar, and Heafield \(2008\)](#) is another clear example of how LDA can be applied to source-code retrieval tasks. In their work they mine various business topics from the source-code of software projects. Their research is in a very typical direction of source-code retrieval, where the goal is to create a system that provides a high-level overview of large legacy software systems to programmers. They focus on identifying the underlying business topics on top of which a particular system have been implemented. The methodology employed in their research is similar to the ones described in Sections [3.2.2](#), [3.2.1](#), [3.2](#): they break down the large body of source-code into source-code documents, create a term-document matrix and, in this case, use the LDA model to derive business topics from the source-code document collection. Their weighting scheme is different from  $tf \times idf$  and is a good example of a domain specific weightage. In their framework the importance factor  $wd[w, f_d]$  for the word  $w$  in the file  $f_d$  is the weighted sum of the frequency of occurrences of  $w$  for each location type  $lt_i$  in the file  $f_d$ .

$$wd[w, f_d] = \sum_{lt_i} \lambda(lt_i) \times v(w, f_d, lt_i)$$

where  $v(w, f_d, lt_i)$  is the frequency of occurrences of  $w$  in location type  $lt_i$  of the file  $f_f$ . Location type here refers to a structural position in the code such as file name, function name, comment. Given the weighted term-document matrix they fit an LDA model on the data and given the of top- $n$  words they manually label the resulting topics. From Apache they extracted 30 topics

and found that the system can successfully model topics spanning through various files and that synonymous words were grouped together in the same topics. They prune the resulting model by using a cutoff value for the probability of words given a topic e.g. if for any given  $w$  given any topic  $t$   $P(w|t) < 0.001$  than  $w$  does not belong to the topic  $t$ . They report that by involving some human intervention - deciding the number of topics, setting the parameters for the importance of location types, validating and labeling topics - their LDA based system can, in fact, provide a good quality overview of large scale systems in terms of implemented business topics.

### 3.3 Source-Code Retrieval as Translation

The method employed in my thesis project is an approach to source-code retrieval introduced by [Deng and Chrupała \(2014\)](#) which treats retrieval as a translation problem. This is different from the various models described in Section 3.2 where the source-code of a particular application is broken down into source-code documents and these documents have their code-internal documentation (e.g. comments) as their features. These source-code documents are then retrieved based on the semantic similarity between their internal documentation and the English query. Under this approach English sentences from the code-internal documentation are treated as pointers to a location in the source-code which is in contrast with the "retrieval as a translation problem" approach employed in my project. Whereas the described approach focuses on retrieving certain parts of a source-code in an application, in my project I focus on retrieving source-code components. A source-code component is a *meaningful* element of a programming language such as loops, conditions, classes or methods. The approach adopted in the Thesis involves learning a translation model from English descriptions to their described Java method-signatures. The translation model does not involve any syntactic analysis it is merely a bag-of-words translation model, encoding associations between terms in separate dictionaries. Learning a bag-of-words translation model aims at providing a grounded model for the meaning of English words where the individual English terms are grounded in the "method-signature-language" space. Other than grounding the goal of the translation model is to enable users to retrieve Java methods from the Java Standard Library given English queries. .

### 3.3.1 Previous Work

I build on the previous work of [Deng and Chrupala \(2014\)](#) where the authors tackle the task of retrieving methods by implementing a language-modelling approach ([Song & Croft, 1999](#)). In their work the rank of method  $m$  is estimated by its probability given the query  $q$ .

$$p(m|q) \propto p(q|m)p(m)$$

They assume a uniform prior for all methods  $m$  and so they rank methods according to  $p(q|m)$ , which under the unigram language model means that:

$$p(q|m) = \prod_{w \in q} p(w|m)$$

Their term-matching baseline model assumes that the queries and method-signatures share the same vocabulary and they set  $p(w|q)$  to the maximum likelihood estimate with Jelinek-Mercer smoothing ([Zhai & Lafferty, 2001](#)):

$$p(q|m) = \prod_{w \in q} (1 - \lambda) f(w|m) + \lambda f(w|M)$$

where  $f(w|m)$  is the relative frequency of the term  $w$  in the method-signature  $m$  and  $f(w|M)$  is the relative frequency of  $w$  in the whole collection of method-signatures  $M$ . The second model in their work is the IBM model 1 where they exploit the fact that descriptions and method-signatures in the Java Standard Library form a parallel-corpus. IBM model 1 is an obvious choice to solve a translation problem as it is one of the simplest translation models available. In IBM model 1 the mapping from description terms to method-signature terms is realized in the form of equioperable alignments between strings; it builds a translation table containing probabilities for every description term  $w$  given a signature term  $u$ :  $p(w|u)$ . Furthermore, the translation probabilities are bootstrapped using expectation maximization and Jelinek-Mercer smoothing is applied to  $p(w|m)$ .

$$p(q|m) = \prod_{w \in q} (1 - \lambda) \left[ \sum_{u \in m} f(w|u) f(u|m) \right] + \lambda f(u|M)$$

The IBM model 1 is not sensitive to initialization of its parameters, which is a major advantage over other similar translation models (Koehn, 2009). Therefore, it is in fact a well suited model for the problem: it is feasible to train, it is designed to find associations between languages and it does provide a translation table for further qualitative analysis. The third and last model they experimented with was the Polylingual Latent Dirichlet Allocation model (Mimno, Wallach, Naradowsky, Smith, & McCallum, 2009). PLDA is designed originally to model relationships between semantically related documents in multiple languages. It is an extension of the LDA model<sup>10</sup>; it organizes document collections in different languages to tuples where documents are about the same - or very similar - topics, but written in different languages. The assumption of PLDA is that every document in a tuple has the same tuple-specific distribution over topics and each topic consists of a set of distributions over words for every language separately. Training the PLDA on the description and method-signature pairs provides distributions of topics over method-signatures  $p(t|m)$  and word distributions of topics  $p(w|t)$ .

$$p(q|m) = \prod_{w \in q} \left[ \sum_{t \in T} p(w|t)p(t|m) \right]$$

They also improve the PLDA model by interpolation using the baseline model.

$$p(q|d) = (1 - \alpha) \times p_{PLDA} + \alpha \times p_{BASELINE}(q|d)$$

Both PLDA and IBM model 1 provide a mapping between vocabularies using the assumption that the Java Standard Library is a parallel corpus between English and Java. Chapter 4 describes my approach in which I take bag-of-words translation as a regression problem and Section 6.1 provides a comparison between the results of the described work and my models.

---

<sup>10</sup>Described in detail in Section 2.3.3

## 4 | **Methods and Data set**

The "retrieval as a translation problem" idea and the data set are both borrowed from the work of (Deng & Chrupała, 2014)<sup>1</sup>. In my project I explore a different approach and attempt to improve their results. Rather than taking a language modelling approach descriptions and method-signatures under my framework are represented in separate vector spaces and the translation model is learned between the pairs of description-signature vectors using supervised regression learning. This method allows for learning highly complex non-linear functions which can potentially result in an improvement over the IBM model 1 and PLDA models. This chapter describes my approach in more detail and I also discuss some details about the implementation of my prototype.

### **4.1 Outline**

This section introduces the data set and the models employed in my research. Section 4.2 introduces the data set and Sections 4.3, 4.4 and 4.5 give short descriptions to the models used for the retrieval system and give some details about their implementation. Section 4.4.3 describes the learning algorithm for the neural network and its hyper-parameters that were taken into account during the project.

### **4.2 Data**

The goal of the project is to learn a translation model from English to Java and to use this model to retrieve Java method-signatures. The data set for this project is the documentation of the Java

---

<sup>1</sup>Lengthy description in Section 3.3.1

Standard Library which conveniently contains method-signatures paired with their descriptions. The idea here is to take these signature-description pairs as a parallel corpus between English and Java to learn a mapping between English terms and Java method-signature terms. Such a mapping allows for a) retrieving Java methods via translation and b) grounding the meaning of English terms in a formal language, namely Java. The Java Standard Library is documented in a format called *Javadoc* and it contains the method name, qualifier, return type, parameter type, names for every method with the addition of a description. The first sentence of the description is a short definition of what the method is used for while the additional sentences might give practical or other details about the method itself. The documentation is broken down into two separate document collections one containing the descriptions and the other containing the signatures. Both document collections went through a pre-processing phase. From the descriptions stop-words and the punctuation were removed, the text was tokenized and all tokens were converted to lowercase. The method-signatures were tokenized in the following way: the method `FilterInputStream` was converted to three terms [filter, input, stream] by breaking up the string using upper cases as borders and then turning the terms into lower case. This process leaves us with two sets of documents each containing a sets of terms (see Figures 4.1 and 4.2). I use the preprocessed data from (Deng & Chrupała, 2014) and it is found online in the public Bitbucket repository of their project: <https://bitbucket.org/gchrupala/codeine>.

### 4.3 Feature vectors

After creating separate document collections for the descriptions and their described method-signatures separate vector-space representations are created from these collections. Dictionaries from both descriptions and signatures are created separately which serve as the basis for the creation of the description-vectors and signature-vectors. This technique allows for numerous options from pruning the dictionary through experimenting with various vector-space representations such as  $tf*idf$ <sup>2</sup> or LSI<sup>3</sup> to various normalizations of the values in the resulting vectors. As a result of this process the two document collections are converted into two matrices: a term-document matrix whose columns represent the descriptions and a term-document

---

<sup>2</sup>See Section 2.3 for more details

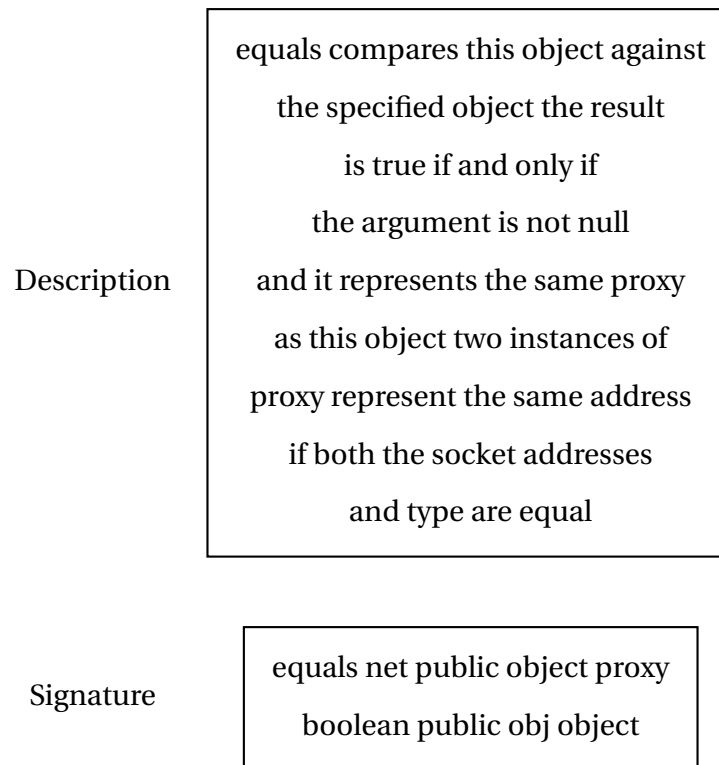
<sup>3</sup>See Section 2.3.2.2 for more details



Figure 4.1: Original Description-Signature pairs

Description	<p>equals Compares this object against the specified object The result is true if and only if the argument is not null and it represents the same proxy as this object Two instances of Proxy represent the same address if both the SocketAddresses and type are equal</p>
Signature	<pre>public final boolean equals(Object obj) package:java.net visibility:public boolean public obj object superclassfulltype:java.lang.Object fulltype:java.net.Proxy method.fulltype:boolean method.visibility:public parameter.name:obj parameter.type:java.lang.Object</pre>

Figure 4.2: Description-Signature pairs after preprocessing



matrix whose columns represent the method-signatures. Each vector-pair in the resulting matrices represents description-method pair from the original Javadoc. This representation makes it possible to learn a bag-of-words translation model by using a regression model which approximates a function that maps description-vectors to their corresponding signatures vector pairs. The resulting function is the bag-of-words translation model from descriptions to signatures which can be used to retrieve signatures given an English query.

## 4.4 Translation Models

In the training phase the above mentioned feature vector representations of the descriptions and the method-signatures are used to train three multivariate regression models. The setting is supervised regression learning where pairs of input description-vectors and target method-signature vectors are presented to the learning algorithm. The vector-space representing descriptions serves as the input matrix and the vector-space representation of signatures serves as the target output matrix for the regression models. The ideal translation model, therefore,

is a function  $T(\mathbf{x})$  that given the description vector  $\mathbf{x}$  produces a signature vector  $\mathbf{s}'$  such that it minimizes the distance  $D$ ,

$$D = \sum_{i=1}^n (\mathbf{s}_i - \mathbf{s}'_i)^2$$

where  $\mathbf{s}_i$  is the target signature vector and  $\mathbf{s}'_i$  is a predicted signature vector.

#### 4.4.1 Linear models

The simplest model to solve a multivariate regression problem is the multivariate linear regression (Craven & Islam, 2011), which assumes that a signature vector is a linear combination of the variables in the description vector

$$\mathbf{s}' = b + w_1x_1 + w_2x_2 + \dots + w_nx_n$$

where  $\mathbf{s}'$  is the predicted signature-vector  $x_1 \dots x_n$  are values in the description-vector,  $b$  is the bias term and  $w_1 \dots w_n$  are coefficients estimated by the model by minimizing the residual sum of squares

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \|X\mathbf{w} - S'\|^2$$

where  $X$  is the matrix of all description-vectors,  $\mathbf{w}$  is the vector of coefficients and  $S'$  is the matrix of all predicted signature-vectors. The issue that can arise when using this simple regression model is over-fitting, meaning that the model perfectly fits the data, but generalizes poorly out of sample. Ridge Regression (Theobald, 1974) penalizes the weights to prevent over-fitting and keeps the coefficients at a moderate size so the model is less able to closely fit the training data. It minimizes the penalized residual sum of squares

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \|X\mathbf{w} - S'\|^2 + \alpha \|W\|^2$$

where  $\alpha \geq 0$  is a hyper-parameter - usually referred to as the *complexity parameter* - that controls the model's robustness against overfitting.

### 4.4.2 Multilayer Perceptron

Both the standard linear regression and Ridge regression fail to model complex non-linear relationships between variables, so I decided to expand the scope of the research to other models that can represent such non-linear functions. One of the best options to solve a highly complex multivariate function approximation problem is the Multilayer Perceptron. It is a type of feed-forward artificial neural network consisting of at least one input, one hidden and one output layer (Haykin, 1994). Each layer consists of several neurons. Given an input vector the values are transformed by the input layer then the hidden layers and finally the output layer. From the hidden layer each hidden neuron  $N$  produces its output as such

$$N_{h_n} = g \left( \sum_{i=1}^n x_i \times w_{hi} \right)$$

where  $N_{h_n}$  is the output of the  $n^{th}$  hidden neuron  $g$  is the non-linear activation function of the neuron,  $x_i$  is the  $i^{th}$  term in the description vector and  $w_{hi}$  is the  $i^{th}$  input weight. The formula is similar to the linear models except a non-linear activation function is "wrapped around" the weighted sum making it possible to approximate sophisticated non-linear functions. The number of parameters to be optimized for a network even with only one hidden layer is significantly larger than the number of parameters of the linear models': The number of parameters of the MLP is  $I \times H + H \times O$  where  $I$  is the number of input neurons,  $H$  is the number of hidden neurons and  $O$  is the number of output neurons. Furthermore, whereas in case of the linear models the error function is convex this is not true for the Multilayer Perceptron. Section 4.4.3 describes a widely used general purpose learning algorithm that was used in the project to train the Neural Network.

### 4.4.3 Training the neural network translation model

As described in Section 4.4 the Neural Translation Model is a function  $f$  that encodes description vector  $\mathbf{x}$  into a signature vectors  $\mathbf{s}$

$$\mathbf{s} = f(\mathbf{x}) = g(W\mathbf{x} + \mathbf{b})$$

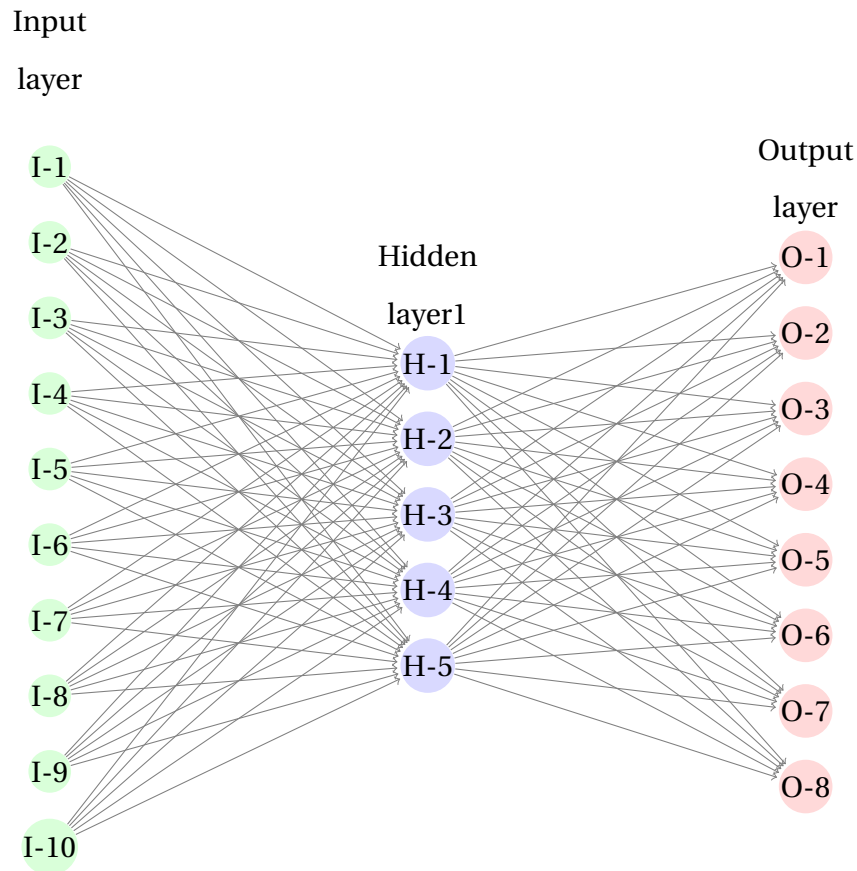
where  $\mathbf{b}$  is the bias and  $W$  is the weight-matrix. The parameters of the model need to be optimized - weights and biases  $\theta = \{W, \mathbf{b}\}$  - through minimizing some cost function  $C(\mathbf{x}_i, \theta)$  where  $\mathbf{x}_i$  is the input vector and  $\theta$  are the parameters. The training algorithm used during the project to minimize the error function is the stochastic gradient descent. The gradient vector of the cost function serves as its basis for computing the parameter updates (Bengio, 2012):  $\frac{\partial L(\mathbf{x}_i, \theta)}{\partial \theta}$ . The most simple version of stochastic gradient descent introduces the hyper-parameter *learning-rate*  $\epsilon$  and utilizes the gradient information and updates parameters  $\theta$  after each training example as

$$\theta^i \leftarrow \theta^{i-1} - \epsilon_i \frac{\partial L(\mathbf{x}_i, \theta)}{\partial \theta}$$

where  $\mathbf{x}_i$  is the training example at iteration  $i$ . Learning rate is considered one of the most important hyper-parameters and to have a large effect on the training. To make convergence easier learning-rate decay is introduced  $\rho$ , another hyper-parameter which decreases the learning-rate at each iteration or after some number of iterations. Although, both hyper-parameters allow for tuning the learning algorithm to the specific problem the direction of the gradient is still only known locally which might not be the true direction, furthermore by searching and moving in the parameter space the true direction of the descent is changing. The idea behind using the original or *mini-batch* version of the stochastic gradient descent is that rather than trying to estimate the gradient direction as close as possible on the whole training set, we do more updates on smaller sets which helps the parameter search to be faster and more comprehensive (Bengio, 2012). Using mini-batches the algorithm averages over the gradients inside each batch  $B$ . In practice using the original stochastic gradient descent (update on one example at a time) or the mini-batch version is much faster than training on the whole training set at once, plus the training does not depend on the size of the data set. The batch-size can be optimized separately from the  $\epsilon$  and  $\rho$  as it only affects the computation time and not the performance of the model on the data sets. However, a slight interaction between  $B$  and  $\rho$  is reported in (Bengio, 2012). To remove oscillations and hesitations of the gradient descent it is common to use a momentum term  $\beta$  - yet another hyper-parameter. To smooth out the stochastic gradient descent samples during the descent a moving average of previous gradients  $\bar{g}$  is computed and  $\beta$  of the importance of the past samples:

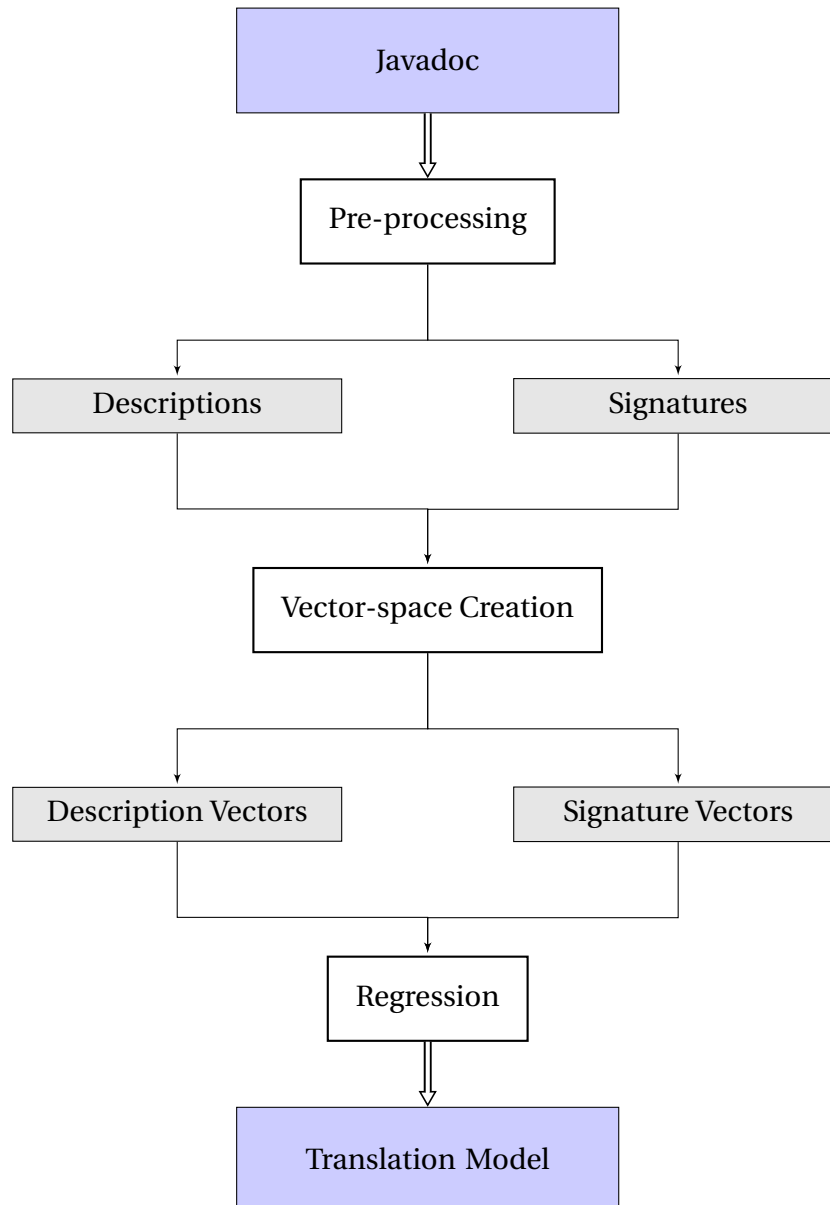
$$\bar{g} \leftarrow (1 - \beta)\bar{g} + \beta g$$

Figure 4.3: Multilayer Perceptron with one Hidden Layer



Similarly to the learning-rate the momentum can be augmented by momentum-decay term  $\alpha$ , leaving us with hyper-parameters  $\Theta = \{\epsilon, B, \rho, \beta, \alpha\}$  which need to be optimized. Optimizing hyper-parameters through exhaustive parameters search was shown to give significant improvements in performance (Pinto, Doukhan, DiCarlo, & Cox, 2009). Hyper-parameters  $\Theta$  are different from parameters  $\theta$  in that they are not learned by the training algorithm, but need to be optimized using some other method. Rather than optimizing hyper-parameters on the whole data set, to get the best values for  $\Theta$  a validation set is used to estimate the impact of the hyper-parameter tuning on the out-of-sample generalization of the model. In consequence the final results need to be reported on yet another data set; the test set.

Figure 4.4: Model Creation



## 4.5 Ranking

The third and final component to the model is the ranking itself. This module receives an English query and transforms it into a query vector according to the vector-space model created from the descriptions. The query-vector serves as the input for the translation model trained on the description-method pairs and it predicts a method-signature-vector from the query. The resulting method-signature vector is compared against all the other method-signature-vectors in the method-signature vector-space and the method -signatures are ranked according to their similarity to the query vector computed by the cosine similarity measure<sup>4</sup>.

---

<sup>4</sup>Described in sections [2.3](#)



## 5 | Experiments

### 5.1 The setup

#### 5.1.1 Data sets

The data set for the experiments is the Java standard library - Standard Edition 6 API Specification: io, lang, math, net, text, util - providing 7183 description-method pairs. Following the traditional machine learning approach the data set is split into three smaller chunks: training set (60%), validation set (20%), test set (20%). The translation models are trained on the training set, the validation set is used for parameter-optimization - to estimate the out-of-sample generalization of the models - and the performance of the models on the test set is presented in the Results section. Only training on the 60% of the data makes it more difficult to obtain good results to report, but allows for a better estimation the models' real-life performance. I follow the split from [Deng and Chrupała \(2014\)](#).

#### 5.1.2 Queries

The performance of the regression models are not evaluated directly on how well they perform the regression itself, but how well they perform in the retrieval tasks. Due to the lack of existing evaluation sets providing English query and Java method-signature pairs, I follow [Deng and Chrupała \(2014\)](#) where validation and test queries are generated from the validation and test sets: queries are the first sentences of the descriptions in the held-out sets. These first sentences originally contain the name of the method as their first term and these terms are removed to make the queries a little more realistic. Arguably this is not the most realistic way of evaluat-

ing the system, however these resulting queries are similar to user queries in that they are short and describe the methods' basic functionality<sup>1</sup>.

## 5.2 Evaluation

The method-signatures relevant to queries are given by the data set; every query has only one relevant method signature as there is a one-to-one correspondence between descriptions and method-signatures in the data set. The models are trained on the description-signature pairs in the training set and tested on the test queries using the whole collection of signatures as candidate answers. This way of ranking a large number of methods makes the evaluation more realistic and the task more challenging. One of the evaluation metrics used to assess the performance of the model is the Mean Reciprocal Rank. This statistic is widely used to evaluate any system that ranks objects according to some similarity measure given a query. It takes the number of queries  $N_q$  and the position in the ranked list of the  $i^{th}$  returned signature to the query  $q^i$ . A set of queries needs to be presented to the trained model and all the method signatures in the collections need to be ranked according to their similarity to the query and the positions of the relevant method signatures to these sample queries needs to be stored. Then the Mean Reciprocal Rank is computed as follows:

$$MRR = \frac{1}{N_q} \sum_{i=1}^{N_q} \frac{1}{rank_i}$$

The other two evaluation metrics reported in the Results section are Accuracy at 1 and Accuracy at 10. These metrics are perhaps more intuitive than the *MRR* in that they answer the question: what proportion of the correct answers were ranked first and in the top 10 respectively.

## 5.3 Vector Representation

As mentioned in Section 4.3 descriptions and signatures are represented in separate vector-spaces. First a dictionary is created for descriptions and signatures separately and both dic-

---

<sup>1</sup>Section 7.1.1 describes an ongoing project whose goal is to develop a more realistic evaluation set.

Table 5.1: The effect of vector representations on the *MRR* on the validation set

	binary	tf	tf*idf ( $L_2$ )	tf*idf ( $L_1$ )
MRR	0.324	0.324	0.333	0.197

The experiments were run by training the Multilayer Perceptron with 800 hidden-neurons and learning-rate: 2.0 on the training set for maximum 29 epochs.

tionaries are pruned using the same method: all terms occurring in more than half of the documents were removed. Removing extremely rare words produced worse results, therefore no pruning was done in that direction. The final version of the dictionaries contain 6299 unique tokens for descriptions and 1717 unique tokens for signatures. Based on the final dictionaries both document collections were transformed into term-document matrices. Various vector representations were considered: binary vectors, term frequency vectors and tf\*idf vectors. The choice of vector-representation did have an effect on the performance of the learning-algorithm. The experiments for finding the best vector-representation were run by training the MLP with constant architecture and hyper-parameters - 800 hidden-neurons, learning-rate: 2.0, max 29 epochs - on the various matrices and evaluating its performance on the validation set. The performance difference was measured on the *MRR*. There was virtually no difference between training the network on binary or term-frequency vectors. The tf\*idf vectors, however, improve the results and the tf\*idf vectors  $L_2$  normalized proved to give the best overall results. Reducing the dimensionality of the feature-spaces using LSI made the training faster, but made the retrieval results only slightly better than random, which is far from desired. In conclusion the vector-spaces were constructed from  $L_2$  normalized tf\*idf vectors with the dimensionality of 6299 for descriptions and 1717 for signatures. The Gensim Python package was used for the vector-space creation tasks (Řehůřek & Sojka, 2010). It is well suited for the task as it has an easy-to-use interface and it provides a wide range of tools for distributional semantics. It proved to be fast enough as for the two document collections of 7183 entries the transformation to two vector-spaces of dimensionality 6299 and 1717 takes only 11.873s running time. The parameters were tuned on the validation set, therefore the results of the experiments on Table 5.1 are also reported on the validation set.

## 5.4 Retrieval engine

Gensim provides a module to calculate the similarity between the query and all the method signatures very efficiently using the cosine similarity measure. It stores all the descriptions and methods in an index which serves as an input to a document similarity method which returns a list of tuples. Each tuple contains the index of the method signature - its identifier -, its similarity to the query and it also provides a ranking as the identifier-similarity tuples are ordered according to similarity. This list of tuples serves as the actual output of the search engine and the evaluation described in Section 5.2 is performed on it.

## 5.5 Parameter tuning for training

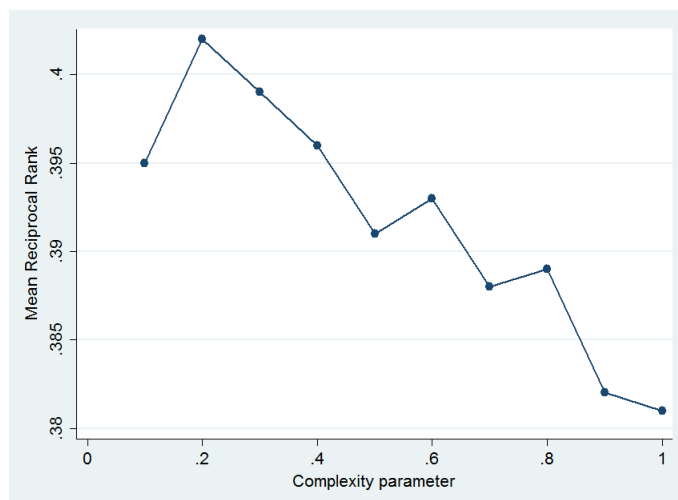
### 5.5.1 Training the Linear Models

The Python package Scikit-learn (Pedregosa et al., 2011) provided both the Linear Regression and Ridge Regression models. As a baseline the simple multivariate Linear Regression was used to approximate a function that maps the description-vectors to signature-vectors. It was the simplest model to work with as it does not have any hyper-parameters. Ridge regression is also an easy-to-work-with model compared to other more sophisticated machine learning algorithms such as SVM. The only hyper-parameter of Ridge is the complexity parameter  $\alpha$ . It was estimated by varying the parameter and evaluating the resulting models on the validation set. Figure 5.1 demonstrates the impact of the various  $\alpha$  parameters on the validation set. The best results were obtained by using Ridge with  $\alpha = 0.2$ . Ridge Regression was a very quick model to train: 1m24.320s.

### 5.5.2 Neural Networks Training

#### 5.5.2.1 Implementations

When training the Neural Network there is an abundance of choices in terms of network architecture, learning algorithms and hyper-parameters, furthermore, there are a number of packages and libraries available for Neural Networks in Python. The obvious choices for a Python li-

Figure 5.1: Impact of  $\alpha$  on *MRR* on the validation set

The experiments to find the best value for  $\alpha$  were run on the training set using  $\text{tf}^*\text{idf}$  vectors and the results are shown on the validation set.

brary that has Neural Networks for Machine learning purposes are PyBrain (Schaul et al., 2010), Theano (Bergstra et al., 2010) and maybe less obvious Theano-Nets<sup>2</sup>. With the linear models the choice for Scikit-Learn was straightforward, but given the number of parameters of a neural network model and the available learning algorithms and hyper-parameters, finding the appropriate library turned out to be a major bottleneck. The first package to be implemented within the project was PyBrain. It was easy to work with and it proved that the MLP is a good choice for solving the problem, but it turned out to be very slow, which made the hyper-parameter tuning infeasible. The two libraries that were more successful were Theano and Theano-nets. Theano is a Python library that is built for optimized compilation of complex mathematical expressions to C code, running on both CPU and GPU. Theano is suitable for building Neural Networks given it's features such as it provides a high-level description language for matrix and tensor operations and performs automatic construction of symbolic graphs to compute gradients. Theano-nets is a library under development that implements the most widely used networks and training algorithms using Theano. Both packages were experimented with dur-

---

<sup>2</sup>The package is still under development and there is no reference available. It can be downloaded from <https://github.com/lmjohns3/theano-nets>

ing the project. In theano-nets the stochastic gradient decent did not seem to converge<sup>3</sup> and the Hessian-free training algorithm was used, but it was too slow and hyper-parameter tuning was again unfeasible. As a result I used a very simple Multilayer Perceptron neural network with stochastic gradient decent implemented in Theano, which produced slightly better results in the end than the experiments with theano-nets. The network was implemented by Grzegorz Chrupala and it is available at <https://bitbucket.org/gchrupala/neuralnet>.

### 5.5.2.2 Network Architecture

The most obvious choice for the architecture was a Multilayer Perceptron with a single hidden layer. This is a model that has been used for decades and therefore all the details regarding its training and hyper-parameters are referred in a large body of literature. It is also a simple model in the world of Neural Networks which makes the learning algorithms perform on it very well as they are designed to handle much more complex architectures as well. Apart from the number of layers the number of neurons in each layer is also a variable. The regressor's number of input neurons equals to the number of values in the input vector - number of terms in the description dictionary - and the number of output neurons equals to the number of values in the target vector - number of terms in the signature dictionary. The number of neurons in the hidden-layer did affect the results when keeping the hyper-parameters and the type of vector-representation constant. Since the  $L_2$  normalized tf\*idf vectors gave the best results, the experiments regarding the number of hidden-neurons were run on tf\*idf term-document matrices. The hyper-parameters for the experiments were the same as for the feature-vector selection experiments: learning-rate = 2.0. max 29 epochs. The final architecture of the network is: 6299 input neurons, one hidden layer with 2500 neurons, 1717 output neurons. The increase in *MRR* from 800 neurons to 2500 is 0.019. Given the time constraints of the project the impact of further increase was not evaluated: The execution time with 800 neurons was 42m45.370s, with 2500 neurons it was 90m17.020s, while 2500 neurons only provided a slight increase in *MRR* (0.001) over 2000 neurons. Table 5.2 shows the results on the validation set for the experiment regarding the number of hidden-neurons. Figure 5.2 shows that the relationship between the

---

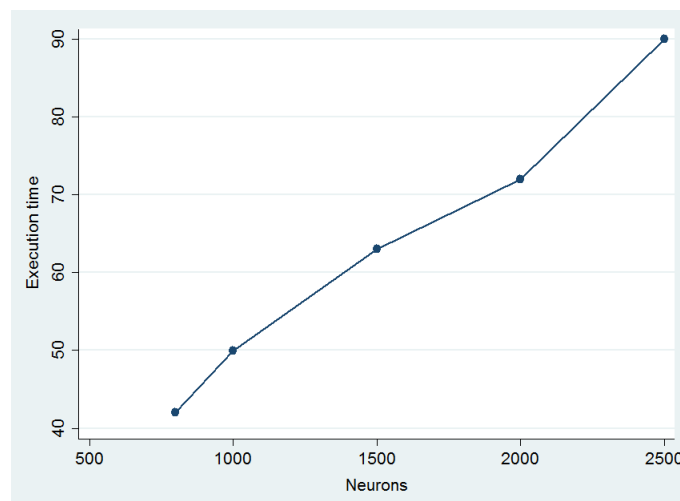
<sup>3</sup>The reason is unclear as it demonstrated no problem on the classification of the iris data set. The bug has been reported and might have already been dealt with. I still strongly recommend theano-nets for experimentation.

Table 5.2: Effect of the number of hidden-neurons on *MRR*

Size of hidden-layer	800	1000	1500	2000	2500
MRR	0.333	0.333	0.342	0.351	0.352
Execution time	42m45.370s	50m25.633s	63m47.261s	72m27.255s	90m17.020s

The Multilayer Perceptron was trained on the training set with learning-rate = 2.0 for maximum 29 epochs. The results are reported on the validations set.

Figure 5.2: Effect of the number of neurons on execution time on the validation set

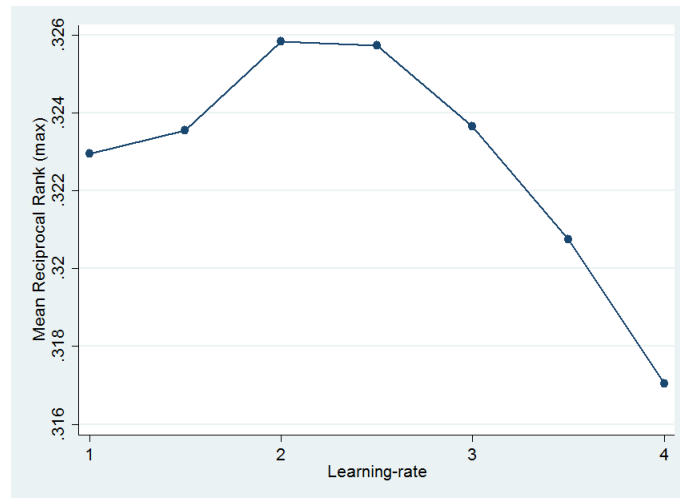


The Multilayer Perceptron was trained on the training set with learning-rate = 2.0 for 29 epochs.

number of neurons in the hidden-layer and the execution time is close to linear.

### 5.5.2.3 Hyper-parameter tuning

As mentioned in Section 5.5.2.1 one of the major bottlenecks for the project was to implement the Neural Network itself. After experimenting with multiple libraries a simple Multilayer Perceptron implemented in Theano provided the best results. The MLP was trained using the Stochastic Gradient Descent algorithm, which is one of the simplest and most general purpose trainers. It is a gradient propagation based algorithm and is explained in detail in Section 4.4.3. The only hyper-parameter that was experimented with in the project was the learning-rate. The experiments to determine the best value for the learning-rate were run on the training set and the performance was evaluated on the validations set. The setup involved tf\*idf vectors, 800

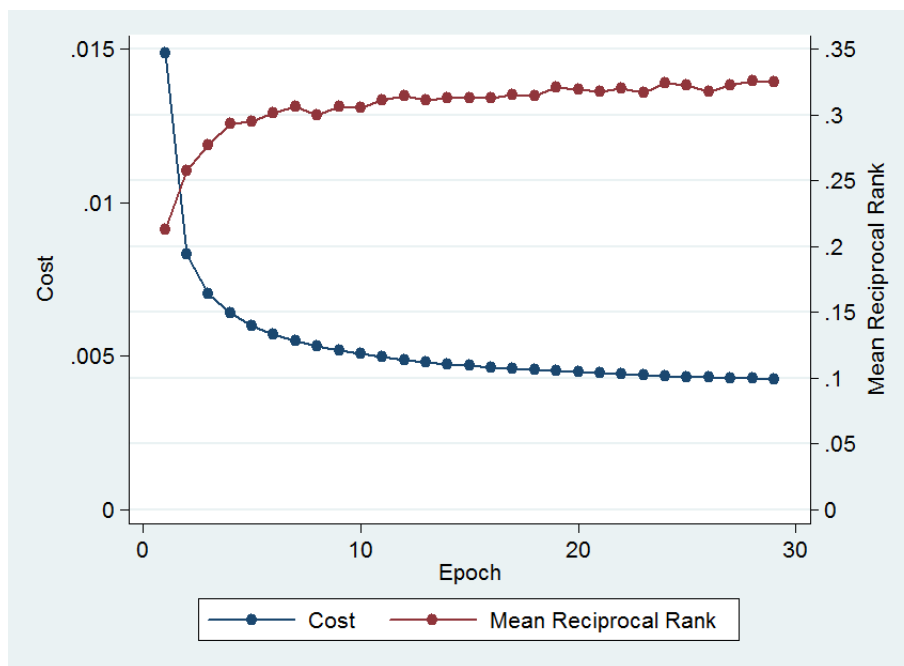
Figure 5.3: Effect of the learning-rate on *MRR* on the validation set

The experiments to determine the best value for the learning-rate were run on the training set with *tf\*idf* vectors, 800 neurons in the hidden-layer and 29 maximum iterations. The maximum values for *MRR* are shown on the graph.

neurons in the hidden-layer and 29 maximum iterations and the best value for the learning-rate was found to be 2.0. Figure 5.3 shows the results of these experiments. The goal of the learning was not to minimize the cost, but to maximize the *MRR*. The cost function of the training-algorithm for this project was the mean squared error. To test how appropriate mean squared error was for the task, for every epoch the value for the cost function and for the *MRR* was stored and analyzed. A significant, strong, negative correlation was found between cost and *MRR* -  $p < .001$   $r = .972$ , with learning-rate 2.0 -, which means that minimizing the cost function does maximize the *MRR*. Figure 5.4 demonstrates the relationship between the cost function and *MRR*.



Figure 5.4: Effect of the learning-rate



The graph shows the *MRR* and the cost function of the MLP when training on the training set and validating on the validation set with *tf\*idf* vectors, 800 neurons in the hidden-layer, learning-rate=2.0 for 29 iterations. The values for the cost function are shown on the *y*-axis to the left and for the *MRR* on the *y*-axis to the right.

## 6 | Results

### 6.1 Results on Retrieval

This section reports the models' evaluation on the retrieval task. As described Section 5.2 the evaluation metrics used to measure retrieval performance are mean reciprocal rank, accuracy at 1 and accuracy at 10. The models are compared against the models in the previous work by [Deng and Chrupala \(2014\)](#)<sup>1</sup>. Linear Regression produced the worst overall results with  $MRR = 0.129$ ,  $Acc@1 = 0.073$  and  $Acc@10 = 0.238$  giving a performance significantly worse than the baseline in [Deng and Chrupala \(2014\)](#). On the other hand Linear Regression performed significantly better than ranking documents at random, which proved that the bag-of-words translation can be phrased as a regression problem. The best performing model was Ridge Regression with complexity parameter  $\alpha = 0.2$ ,  $MRR = 0.389$ ,  $Acc@1 = 0.232$  and  $Acc@10 = 0.707$ . On both  $MRR$  and  $Acc@10$  the model outperforms the term-matching baseline and PLDA from [Deng and Chrupala \(2014\)](#) and almost matches it on  $Acc@1$  - worse with 0.1. The Multilayer Perceptron produced comparable results to Ridge Regression-  $MRR = 0.341$ ,  $Acc@1 = 0.202$ ,  $Acc@10 = 0.640$  -, but whereas Ridge Regression clearly beats PLDA, the Multilayer Perceptron performs even worse on  $Acc@1$  than the Ridge Regression. However, the MLP does have comparable performance as the PLDA on  $MRR$  and has a better  $Acc@10$ . In conclusion, none of the models developed in my thesis project managed to perform better than the IBM model 1 in [Deng and Chrupala \(2014\)](#), both the MLP and Ridge Regression gave better  $Acc@10$  and  $MRR$  than the PLDA and Ridge Regression outperformed the PLDA overall. Results are shown in Table 6.1 comparing the results of the present thesis with the results of [Deng and Chrupala \(2014\)](#).

---

<sup>1</sup>Decribed in Section 3.3.1

Table 6.1: Comparison of the models on the validation set

Results from the Thesis			
Model	MRR	Acc@1	Acc@10
Linear Regression	0.129	0.073	0.238
Ridge Regression( $\alpha = 0.2$ )	<b>0.389</b>	<b>0.232</b>	<b>0.707</b>
Multilayer Perceptron( $\epsilon = 2.0$ )	0.341	0.202	0.640
Term-matching	0.332	0.223	0.530
PLDA	0.352	0.242	0.562
IBM model 1	<b>0.493</b>	<b>0.339</b>	<b>0.793</b>

Above are the models employed in the thesis and below the models from [Deng and Chrupala \(2014\)](#). All models were trained on the training set using tf\*idf vectors and the results are reported on the test set. The Multilayer Perceptron with 2500 hidden neurons was trained with maximum iterations of 29.

## 6.2 Word Similarity

After successful training the model is able to retrieve method signatures give English queries, but also to ground the meaning of English terms. To test how well the model captures the meaning of the English terms, the words in the descriptions are embedded in the space of the hidden layer of the network. This involves creating separate vectors for every word in the description dictionary - vectors with 6298 zeros and 1 one value - and feeding the resulting matrix to the model and retrieving the transformation from the hidden layer. A matrix containing the translated word-vectors is the result of this process. By calculating the pairwise cosine similarity for all word-vectors with all other vectors we get a sense of how well semantic relatedness between words have been captured by our model. There was no quantitative analysis of the success of the model's grounding performance. Table 6.2 shows the word-similarity lists obtained by the method explained in this section. Table 6.3 demonstrates that date-related terms are clustered together after this translation. Hopefully the tables give the impression to the reader that the meaning of English words is captured by the model with examples such as: (true, tests, whether, boolean, false). These results demonstrate that the bag-of-words translation approach to Source Code Component retrieval can be used to ground the meaning of natural language with Java-

Table 6.2: Word Similarity in the hidden layer

zip	currency	http	true	scale	cos
compressed	symbol	cookie	whether	preferred	argument
compression	countries	connection	boolean	rounding	trigonometric
checksum	currency	registers	equality	true	cosine
defaultcompression	represent	urlconnection	switching	decimal	angle
uncompressed	territories	proxy	false	big	oneargument

Table 6.3: Time related terms

timezone	zones	sep	gmstring	yyyy
eastern	west	oct	gmtstring	sep
thisget	utcthisget	nov	sep	oct
zones	massachusetts	mm	oct	nov
west	eastern	jun	nov	mm
utcthisget	sun s	jul	jun	jun

terms using distributional methods.

### 6.3 Prototype

I have created a simple prototype for my thesis, which is a user interface for the search engine where the user can give an English query and the prototype returns a list of Java method-signatures from the Java Standard Library along with their descriptions. It is intended to test and demonstrate what the evaluation scores mean in a real-life application. I created it so that in further research possibly students can use the prototype to evaluate the system with real-life users. The prototype uses the Ridge Regression model as it provided the best results from the models implemented in my thesis. There was an experiment conducted if it would be beneficial to implement a simple spelling corrector to handle potential typos in the prototype. The experiments were run using Peter Norvig's spelling corrector example from <http://norvig.com/spell-correct.html>. It is a very simple spelling corrector and was trained on the description

Figure 6.1: Searchbox with link to the repo

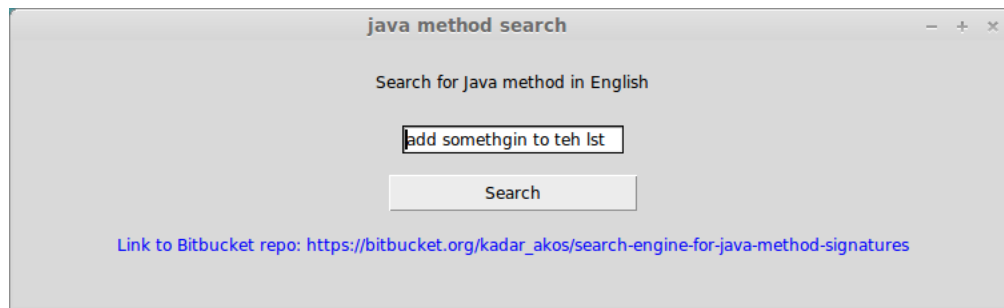
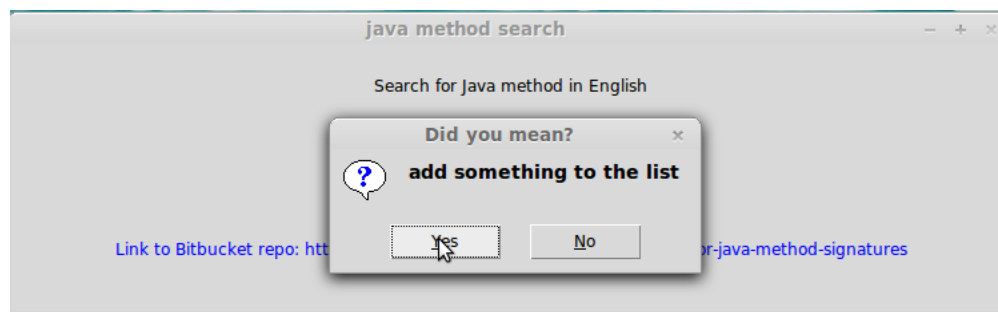
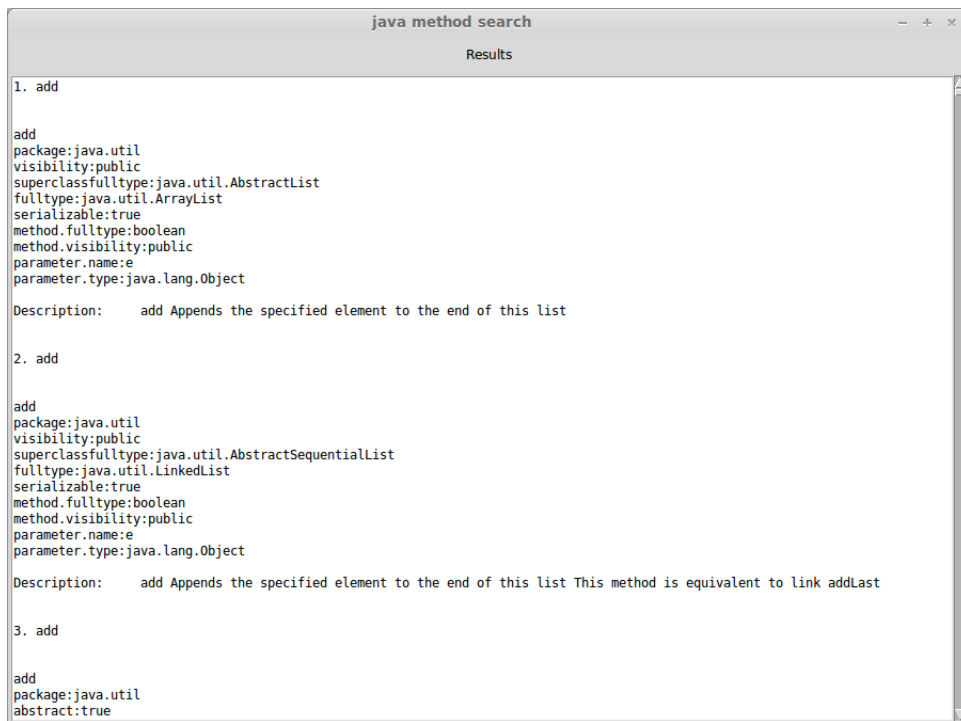


Figure 6.2: Dialogue box with the suggested query



document collection and it successfully corrected some simple test typos such as "add somethgin to teh lst" to "add something to the list". To test whether the spelling corrector wrongfully corrects correct queries, the validation and test queries were concatenated and ran through the spelling corrector. It was found that 9.34% - 224 out of 2396 - were wrongfully corrected and therefore by default the corrector is disabled and it can be enabled optionally. If the spelling corrector is enabled and the users query and the suggested query are not equal, a dialog box asks whether the user would like to query with the original query or with the suggested one. The intention behind implementing the spelling corrector was only to provide a more complete prototype for my thesis project. All the data files and scripts to run the experiments can be found in [https://bitbucket.org/kadar\\_akos/search-engine-for-java-method-signatures](https://bitbucket.org/kadar_akos/search-engine-for-java-method-signatures). Running GUI.py runs the prototype. The two screens and the dialogue box of the interface are shown in Figures 6.1, 6.2 and 6.3.

Figure 6.3: Search results



The screenshot shows a window titled "java method search" with a "Results" tab. It displays three search results for the "add" method, each with its metadata and a description.

```
1. add
add
package:java.util
visibility:public
superclassfulltype:java.util.AbstractList
fulltype:java.util.ArrayList
serializable:true
method.fulltype:boolean
method.visibility:public
parameter.name:e
parameter.type:java.lang.Object
Description:    add Appends the specified element to the end of this list

2. add
add
package:java.util
visibility:public
superclassfulltype:java.util.AbstractSequentialList
fulltype:java.util.LinkedList
serializable:true
method.fulltype:boolean
method.visibility:public
parameter.name:e
parameter.type:java.lang.Object
Description:    add Appends the specified element to the end of this list This method is equivalent to link addLast

3. add
add
package:java.util
abstract:true
```

## 7 | Future work and Conclusion

### 7.1 Future work

#### 7.1.1 More realistic evaluation

The objective of the project *Mining specialized knowledge from online communities* at Tilburg University is to build a data set consisting of pairs of questions and relevant answers to evaluate the models on a more realistic data set. The corpus is built by choosing appropriate questions from Stackoverflow and annotating the structure of the answers. It was an obvious decision to choose Stackoverflow as a source of relevant data for the project: It is a large Q&A website dedicated to questions and answers in a wide variety of topics related to computer science and especially to programming. The annotation process for the project is designed to collect relevant question-answer pairs to evaluate the method retrieval prototype. The annotators input a filter in the Stackoverflow search-box - to filter out irrelevant questions - and open questions one-by-one. For each of the questions they decide which are the relevant answers and from each relevant answer they identify the relevant methods. The result of the annotation process is a data set consisting of pairs of real-life questions and relevant methods. This is an ongoing project and so far annotators store the id for every question and answers they find relevant and the names and classes of the methods. Other necessary information for the method-signature retrieval is missing at this stage such as number of arguments or argument types. This method-signature resolution part is currently unfinished, therefore in the present thesis I am not able to use this evaluation set and it is subject to future work. Source-code, data and details for the project are found on [https://bitbucket.org/kadar\\_akos/data-collection](https://bitbucket.org/kadar_akos/data-collection)<sup>1</sup>.

---

<sup>1</sup>The repository for the project is public, but please note that it is an ongoing and unfinished project.

### 7.1.2 Extensive tuning of the Multilayer Perceptron

During my project the possibility of using regression models for Source Code Component Retrieval was explored and the results were promising. The models match the baseline in [Deng and Chrupała \(2014\)](#) and even outperform the PLDA model. From the results it is clear that the MLP is a potential candidate to solve the problem. However, it was outperformed by Ridge Regression, which is due to the fact that its error function is convex, while for the MLP it has a number of local minima. In [Section 5.5.2.1](#) I mention that the implementation of the Neural Network was a large bottleneck and consequently one of the shortcomings of the project was that the parameter optimization on the MLP was not extensive enough. The network architecture in my project was fixed to a Multilayer Perceptron with a single hidden-layer and the impact of more hidden-layers has not been tested. Furthermore, the only hyper-parameter that was used during my project was the learning-rate. The fact that Ridge Regression outperforms Linear Regression draws to the conclusion that one of the most important hyper-parameter that should be experimented within further work for the MLP is the  $L_2$  regularization of weights. Other important hyper-parameters include learning-rate-decay, momentum and momentum-decay. All of the mentioned hyper-parameters should be experimented within further work, preferably using wide coverage grid-search. If trained properly the MLP has the potential to outperform Ridge Regression as it is able to model more complex non-linear functions. [Section 4.4.3](#) I introduces the set of hyper-parameters I would have used given more time.

### 7.1.3 Web-interface

I have created a prototype search engine with a graphical user-interface that retrieves Java method-signatures and their descriptions given an English query using Ridge Regression. It is intended to help future researchers to evaluate the various retrieval models on real-life users. It would be beneficial to develop a web-interface for the search engine to enable researchers to evaluate the model on a larger number of users. The development of such an interface is subject to future work.



## 7.2 Conclusion

The main goal of my project was to develop new models to perform source-code component retrieval. I based my research on the work of (Deng & Chrupała, 2014) where the authors use Polylingual Latent Dirichlet Allocation and IBM model 1 to learn a bag-of-words translation model between Java-method signatures and English descriptions and use these models to retrieve method-signatures given English queries. My aim was to take a different path and use regression models as bag-of-words translation models and hopefully outperform the models presented in (Deng & Chrupała, 2014). The other goal was to evaluate how well these models capture the meaning of English words. The system developed during my thesis project manages to outperform the baseline and PLDA from (Deng & Chrupała, 2014). The Ridge Regression outperforms PLDA overall evaluation measures. The Multilayer Perceptron have comparable result to PLDA on *MRR* and beats it on *Acc@10*. Although, both models perform decently and the project can be considered a success in that regard, they do not beat the IBM model 1 from (Deng & Chrupała, 2014). The other main aim was to test whether the models ground the meaning of English expressions in Java method-signature terms. To test if this was successful a word-similarity experiment was conducted using the Multilayer Perceptron. It was found that both models provide intuitively good results in grounding. Furthermore, the research project has proved that learning a bag-of-words translation model is possible using general purpose regression-models widely used in many fields of Machine Learning. Furthermore, a prototype was created, which can be used to retrieve method-signatures given English queries and it can found on [https://bitbucket.org/kadar\\_akos/search-engine-for-java-method-signatures](https://bitbucket.org/kadar_akos/search-engine-for-java-method-signatures).

# References

- Andreas, J., Vlachos, A., & Clark, S. (2013). Semantic Parsing as Machine Translation.
- Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the Trade* (pp. 437–478). Springer.
- Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., . . . Bengio, Y. (2010, June). Theano: a CPU and GPU math expression compiler. In *Proceedings of the python for scientific computing conference (SciPy)*. (Oral Presentation)
- Berry, M. W., & Browne, M. (1999). *Understanding search engines: mathematical modeling and text retrieval* (Vol. 8). Siam.
- Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *the Journal of machine Learning research*, 3, 993–1022.
- Cai, J. F., Lee, W. S., & Teh, Y. W. (2007). NUS-ML: Improving word sense disambiguation using topic features. In *Proceedings of the 4th International Workshop on Semantic Evaluations* (pp. 249–252).
- Chen, D. L., & Mooney, R. J. (2008). Learning to sportscast: a test of grounded language acquisition. In *Proceedings of the 25th international conference on Machine learning* (pp. 128–135).
- Cleary, B., & Exton, C. (2006). The Cognitive Assignment Eclipse Plug-in. In *ICPC* (pp. 241–244).
- Craven, B., & Islam, S. M. (2011). *Ordinary least-squares regression*. SAGE Publications.
- Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., & Harshman, R. A. (1990). Indexing by latent semantic analysis. *JASIS*, 41(6), 391–407.
- Deng, H., & Chrupała, G. (2014). Semantic approaches to software component retrieval with English queries. *LREC*.
- Dit, B., Revelle, M., Gethers, M., & Poshyvanyk, D. (2013). Feature location in source code: a

- taxonomy and survey. *Journal of Software: Evolution and Process*, 25(1), 53–95.
- Dumais, S. T., Furnas, G. W., Landauer, T. K., Deerwester, S., & Harshman, R. (1988). Using latent semantic analysis to improve access to textual information. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 281–285).
- Gay, G., Haiduc, S., Marcus, A., & Menzies, T. (2009). On the use of relevance feedback in IR-based concept location. In *Proceedings of the ICsM 2009. IEEE International Conference on software Maintenance* (pp. 351–360).
- Glenberg, A. M., & Kaschak, M. P. (2002). Grounding language in action. *Psychonomic bulletin & review*, 9(3), 558–565.
- Harnad, S. (1990). The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1), 335–346.
- Harnad, S. (2003). Symbol-grounding problem. *Encyclopedia of cognitive science*.
- Harris, Z. S. (1954). Distributional structure. *Word*.
- Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.
- Koehn, P. (2009). *Statistical machine translation*. Cambridge University Press.
- Kuhn, A., Ducasse, S., & Gírba, T. (2007). Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49(3), 230–243.
- Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 104(2), 211.
- Lenci, A. (2008). Distributional semantics in linguistic and cognitive research. *Italian journal of linguistics*, 20(1), 1–31.
- Liu, C., Yan, X., Fei, L., Han, J., & Midkiff, S. P. (2005). SOBER: statistical model-based bug localization. *ACM SIGSOFT Software Engineering Notes*, 30(5), 286–295.
- Lukins, S. K., Kraft, N. A., & Etzkorn, L. H. (2008). Source code retrieval for bug localization using latent dirichlet allocation. In *Proceedings of the WCrE, 2008. 15th Working Conference on Reverse Engineering*. (pp. 155–164).
- Lund, C. B., & Kevin. (1997). Modelling parsing constraints with high-dimensional context space. *Language and cognitive processes*, 12(2-3), 177–210.
- Marcus, A., & Maletic, J. I. (2001). Identification of high-level concept clones in source code.

- In *Proceedings of aSE 2001. 16th annual International Conference on Automated Software Engineering* (pp. 107–114).
- Marcus, A., Rajlich, V., Buchta, J., Petrenko, M., & Sergeyev, A. (2005). Static techniques for concept location in object-oriented code. In *Proceedings of the IWpC 2005. 13th International Workshop on program Comprehension* (pp. 33–42).
- Marcus, A., Sergeyev, A., Rajlich, V., & Maletic, J. I. (2004). An information retrieval approach to concept location in source code. In *Reverse engineering, 2004. proceedings. 11th working conference on* (pp. 214–223).
- Maskeri, G., Sarkar, S., & Heafield, K. (2008). Mining business topics in source code using latent dirichlet allocation. In *Proceedings of the 1st India software engineering conference* (pp. 113–120).
- Materna, J. (2012). LDA-Frames: an unsupervised approach to generating semantic frames. In *Computational Linguistics and Intelligent Text Processing* (pp. 376–387). Springer.
- Mathe, S., Fazly, A., Dickinson, S., & Stevenson, S. (2008). Learning the abstract motion semantics of verbs from captioned videos. In *Computer Vision and Pattern Recognition Workshops, 2008. cVPRW'08. IEEE computer Society conference on* (pp. 1–8).
- Matuszek, C., Fox, D., & Koscher, K. (2010). Following directions using statistical machine translation. In *Proceedings of the 5th ACM/IEEE international conference on Human-robot interaction* (pp. 251–258).
- Mimno, D., Wallach, H. M., Naradowsky, J., Smith, D. A., & McCallum, A. (2009). Polylingual topic models. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language processing: Volume 2-Volume 2* (pp. 880–889).
- Mooney, R. J. (2007). Learning for semantic parsing. In *Computational Linguistics and Intelligent Text Processing* (pp. 311–324). Springer.
- Mooney, R. J. (2008). Learning to Connect language and Perception. In *AAAI* (pp. 1598–1601).
- Nevin, B. E. (2002). *The Legacy of Zellig Harris: Language and information into the 21st century. Volume 1: Philosophy of science, syntax and semantics* (Vol. 228). John Benjamins Publishing.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*,

12, 2825–2830.

- Pinto, N., Doukhan, D., DiCarlo, J. J., & Cox, D. D. (2009). A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS computational biology*, 5(11), e1000579.
- Poshyvanyk, D., & Marcus, A. (2007). Combining formal concept analysis with information retrieval for concept location in source code. In *Program Comprehension, 2007. ICpC'07. 15th IEEE International Conference on* (pp. 37–48).
- Řehůřek, R., & Sojka, P. (2010, May 22). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (pp. 45–50). Valletta, Malta: ELRA.
- Richardson, K., & Kuhn, J. (2014). UnixMan Corpus: A Resource for Language Learning in the unix Domain. *LREC*.
- Roy, D. (2005). Semiotic schemas: A framework for grounding language in action and perception. *Artificial Intelligence*, 167(1), 170–205.
- Schaul, T., Bayer, J., Wierstra, D., Sun, Y., Felder, M., Sehnke, F., ... Schmidhuber, J. (2010). PyBrain. *Journal of Machine Learning Research*, 11, 743–746.
- Singhal, A. (2001). Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4), 35–43.
- Song, F., & Croft, W. B. (1999). A general language model for information retrieval. In *Proceedings of the eighth international conference on Information and knowledge management* (pp. 316–321).
- Theobald, C. (1974). Generalizations of mean square error applied to ridge regression. *Journal of the Royal Statistical Society. Series B (Methodological)*, 103–106.
- Wong, Y. W., & Mooney, R. J. (2006). Learning for semantic parsing with statistical machine translation. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics* (pp. 439–446).
- Zhai, C., & Lafferty, J. (2001). A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 334–342).