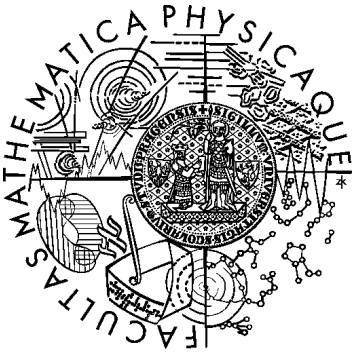
Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS



Maria Ximena Gutierrez Vasques

Quantifying Determiners from the Distributional Semantics View

Institute of Formal and Applied Linguistics

Erasmus Mundus European Master in Language & Communication Technologies (LCT)

Supervisors of the master thesis:	doc. RNDr. Markéta Lopatková, Ph.D. Dr. Raffaella Bernardi.
Study programme:	Informatika
Specialization:	Mathematical Linguistics

Prague 2012

Acknowledgements

I would like to thank to the European Commission that through the Erasmus Mundus program has granted me with the opportunnity of studying a master program in the field of language technologies. This academic and cultural experience has been invaluable.

I'm grateful with the coordinators of the LCT program and with the people working at Charles University and Free University of Bozen-Bolzano. During these two years they have been very helpful providing me the academic and administrative orientation to be able to finish this master in two different universities.

This thesis would not have been possible without the constant support of my supervisors, Dr. Marketa Lopatkova and Dr. Raffaella Bernardi. I should also thank to Dr. Marco Baroni (COMPOSES project), to Dr. Andrea Passerini (University of Trento) and to Dr. Floriano Zini (University of Bozen-Bolzano).

Finally I would like to thank to my family and friends in Mexico and to those friends I've made during these two years living abroad. They have been a support and a companion in the good and bad days.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague date 07. December 2012 signature of the author

Abstract

Název práce: Quantifying Determiners from the Distributional Semantics View

Autor: Maria Ximena Gutierrez Vasques

Katedra: Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce: doc. RNDr. Markéta Lopatková, Ph.D.

Abstrakt: Distribuční sémanika představuje moderní přístup k zachycení sémantiky přirozeného jazyka. Jedním z témat, kterým zatím v rámci tohoto přístupu nebyla věnována dostatečná pozornost, je možnost automatické detekce logických relací jako vyplývání. Tato diplomová práce navazuje na práci autorů Baroni, Bernardi, Do and Shan (2012), kteří se zabývají relací vyplývání mezi kvantifikujícími výrazy. Citovaná práce využívá detekce pomocí SVN klasifikátorů natrénavaných na sémantických vektorech reprezentujících relaci vyplývání. Popisované experimenty se nezaměřovaly na nastaveni parametrů SVN klasifikátoru, proto se v této práci vracíme k původním experimentům popisujícím relaci vyplývání mezi kvantifikovanýmo jmennými konstrukcemi, navrhujeme nové konfigurace klasifikátoru a optimalizujeme nastavení parametrů. Dosaženou přesnost predikce porovnáváme s původními výsledky a ukazujeme, že SVM klasifikátor s kvadratickým polynomiálním jádrem dosahuje lepších výsledků. Analyzujeme úspěšnost výsledků a navrhujeme vysvětlení, proč jsou některé kombinace nastavení úspěšné a co tato úspěšnost odhaluje o zpracovávaných datech. Závěrečné experimenty potvrzují hypotézu, podle které každá sémantická doména vyžaduje vlastní model relace vyplývání.

Klíčová slova: Distribuční sémanika, SVN (support vector machines) klasifikátor, vyplývání, kvantifikující výrazy.

Title: Quantifying Determiners from the Distributional Semantics View

Author: Maria Ximena Gutierrez Vasques

Department: Institute of Formal and Applied Linguistics

Supervisor: doc. RNDr. Markéta Lopatková, Ph.D., Charles University in Prague

Abstract: Distributional semantic models are an approach to natural language semantics that has become popular due to its capacity to successfully capture semantic relations in a relative easy way. One aspect that has been explored little is the use of distributional semantic representations to automatically detect logical relations such as entailment between words or phrases. This thesis starts from the work of Baroni, Bernardi, Do and Shan (2012) that addresses for the first time the entailment between quantifying determiners using distributional semantic models. In the mentioned work, the entailment detection is done by means of an SVM classifier trained with semantic vectors representing pairs of quantifying phrases that are in an entailment relation. The original experiments paid little attention to the parameters involved in an SVM classifier. We repeat the experiments to detect entailment between quantifier-noun constructions, proposing new configurations of the SVM classifier and performing parameter optimization. We compare the prediction accuracy and show that an SVM classifier with a quadratic polynomial kernel is more suitable for the task. We analyze why some combinations of parameters are more successful and what does this reveal about the dataset and the entailment relations. Finally, we run an experiment to reinforce the evidence, from the distributional semantics perspective, that each semantic domain has its own entailment relation.

Keywords: Distributional semantics, support vector machines, entailment, quantifiers.

Contents

Chapter 1

Introduction

1.1 Introduction

In general, we understand semantics as a study of the meaning of a word, a phrase, a sentence. This fundamental question of what is meaning has been tackled by philosophers, linguists and others in the past and recent times. On one hand, linguists and logicians have constructed various highly formalized models in order to be able to describe natural languages as precisely as possible (formal semantics). On the other hand, philosophers as Wittgenstein avoided the question of what is meaning and focused more on the practice of language, suggesting that the meaning of linguistic signs is its use within a context: "Do not look for the meaning, look for the use" (Wittgenstein, 1953).

Distributional Semantics (DS) is an approach to semantics based on the above mentioned notion (Wittgenstein's), where meaning is the family of words or expressions to which a word is similar to. In DS, the meaning of a word is represented as a vector that codes the pattern of co-occurrence of that word with other words in a large corpus. This approach has been proved to be useful for measuring the semantic similarity of words and has many applications like word clustering, word classification, automatic thesaurus generation, word sense disambiguation, etc.

Semantics deals with representing not only the lexical meaning of a word, but also of expressions larger than words. In this sense, formal semantics build a logical representation of the meaning of phrases and sentences by combining the meaning of its constituents. This key notion in semantics is known as compositionality. Recently, DS has also tried to model the meaning of a phrase by composition, some of the approaches perform algebraic operations between

semantic vectors of the words constituting a phrase.

DS has mainly focused on content words (nouns, adjectives, verbs), other types of words like determiners are usually ignored. The determiners, specifically the quantifying determiners or quantifiers (e.g., all, some), play a very important role in formal semantics since they represent logical operators. This thesis has special interest in the work of Baroni, Bernardi, Do and Shan (2012), which explores for the first time the DS representation of quantifier phrases (phrases built from a quantifier and a noun), suggesting that semantic vectors are able to capture the semantic properties not only of content words but also of quantifiers. The evidence show that it is possible to capture and generalize logical inference patterns, entailment, by using the semantic vectors of expressions such as quantifier phrases. Baroni, Bernardi, Do and Shan (2012) are able to automatically detect that there is an entailment relation between the quantifier phrases "all cats" and "several cats" by training a machine learning classifier with examples of semantic vectors representing pairs of quantifier phrases that are in an entailment/non-entailment relation.

Phenomena like quantification and inference have been deeply studied by formal semantics. Analyzing these phenomena from the DS perspective constitutes a bridge between these two aproaches of natural language semantics. This joint view can be helpful to cope with the limitations of each since formal semantics models are usually not enough to deal with the complexity of natural language, while DS models are able to obtain semantic representations extracted from corpora but have focused so far on representing lexical meaning, paying little attention to logical issues.

1.2 Goal

This work starts from the evidence shown in Baroni, Bernardi, Do and Shan(2012) that entailment relations can be detected using DS representations of phrases. We first provide a theoretical background and describe the related work to detect entailment using DS representations.

We are mainly interested in the experiments conducted for detecting entailment between quantifier phrases (QPs), this is, the entailment between quantifier-noun constructions sharing the same noun (e.g.,*many dogs* $\models$ *some dogs*). The semantic vectors encoding the QP's seem to carry enough information allowing a support vector machine (SVM) classifier to predict if there is an entailment relation between two QP's, even if the involved quantifiers were never seen

during the training phase of the classifier. Baroni, Bernardi, Do and Shan (2012) show that an SVM classifier trained with semantic vectors represent a successful entailment recognizer but the analysis of why is this happening is left as future work. The general goal of this thesis is to complement the previous work, we will analyze the machine learning method used and repeat the experiments trying new classifier configurations. We are interested not only in improving the performance of the current entailment detector but in understanding better the properties of the semantic vectors that allow to learn and generalize the entailment relations.

This thesis will look in more depth at the support vector machines since the original entailment-detection experiments used an SVM classifier with a polynomial kernel of degree 3 and a combination of default parameters provided by the used software. We aim to repeat the experiments with an optimized combination of parameters and with different degrees of polynomial kernels (later we will talk about the meaning and impact of these parameters in a classification task). Some of the reasons to do this is that SVM classifiers with lower degree kernels are less complex and make easier to discover which features of the semantic vectors were more relevant or were taken into account when building the learning model, also the type of classifier that is able to obtain the highest accuracies can reveal which kind of relation (linear, non-linear) exists between the pairs of semantic vectors that allows to classify them as entailment or non-entailment pair.

Comparisons will be performed among the performance obtained using different SVM classifiers. We may observe that some classifiers are more accurate when predicting the entailment of certain pairs of quantifiers. From this and from analyzing the detailed predictions we would like to detect some patterns and interpret why some combination of parameters result better or worst and what does that mean in terms of detecting entailment using distributional vectors. The general idea is to be able to perform quantitative analysis but also qualitative by giving a linguistic interpretation to the results produced by the machine learning classifier.

The last experiment will be performed to complement the idea that the entailment relations are different depending on the semantic domain (e.g., nouns, quantifiers). i.e., we will ask the question whether the model that detects entailment between quantifiers is able to generalize to the noun case.

1.3 Thesis structure

In the previous sections we have briefly mentioned some concepts needed to introduce the topic and state the goal of this work. Chapter 2 contains a more complete view and explanation of these concepts, the intention of the chapter is to provide background of the main notions related with distributional semantic models, entailment and machine learning classifiers. These notions are necessary to understand Chapter 3 that contains the related work of entailment from the distributional semantics view and a detailed description of the experiments performed in Baroni, Bernardi, Do and Shan(2012) which constitute the basis of this thesis. Chapter 4 analyzes the machine learning technique that is going to be used for the experiments and explains the reasons why we decided to try different SVM classifiers. The obtained results and the interpretation are also discussed in this chapter. Finally, chapter 5 contains the conclusions and future work.

Chapter 2

Background

2.1 Distributional semantic models

Distributional semantic models (DSMs) – also known as "word space", "distributional similarity" or "corpus-based semantic" models – are an approach to semantic where the meaning of a word is represented by a vector that codes the pattern of co-occurrence of that word with other expressions in a large corpus. These models rely on the distributional hypothesis claiming that words that occur in similar contexts tend to have similar meanings (Wittgenstein, 1953; Harris, 1954; Weaver, 1955; Firth, 1957; Deerwester, Dumais, Landauer, Furnas, & Harsh-man, 1990). DSMs have been popular in the recent years due to their capacity of extracting knowledge automatically from a given corpus, requiring much less labour than other approaches to semantics, such as hand-coded knowledge bases and ontologies (Turney and Pantel, 2010).

DSMs are based on vector space models, this kind of models are widely used in information retrieval. The main idea is to represent each document in a collection as a point in a space (a vector in a vector space). Points that are close together in this space are semantically similar and points that are far apart are semantically distant. In a DSM, a word is represented by a vector in which the elements are derived from occurrences of that word in various contexts. The contexts may be windows of words sorrounding the target word (Lund & Burgess, 1996), entire paragraphs or documents (Landauer and Dumais 1997; Griffiths, Steyvers, and Tenenbaum 2007), grammatical dependencies (Lin, 1998; Pado & Lapata, 2007), and some other richer contexts. The matrix formed by distributional vectors is usually known as semantic space or word-context matrix. This matrix collects the co-occurrence information, each row corresponds to a target word and each column represents a given linguistic

context. The idea is that similar row vectors indicate similar word meanings, the semantic similarity can be measured using a distance measure such as cosine or Euclidean distance.

A DSM can be defined as a scaled and/or transformed co-occurrence matrix M, such that each row x represents the distribution of a target term across contexts, the columns are the set of elements representing the contexts used to compare the distributional similarity of the target terms. A target term (or term of interest) may be a word, lemma, phrase, morpheme. If we have a $m \times n$ co-occurrence matrix M, then the m rows are the target terms and the n columns are the features or dimensions. Some of the parameters and steps that must be taken into account when building a DSM, include (Baroni and Lenci 2010; Turney and Pantel, 2010):

Corpus pre-processing: In this first step, the terms for building the DSM are identified. The corpus must be tokenized (segmented in smaller units, usually words). Some deeper linguistic analysis may be performed such as part-of-speech tagging, lemmatization, stemming, shallow syntactic patterns, dependency parsing.

Type of DSM: This distinction between DSMs relies on the representation of the co-occurrence relations between a target term and a context. Unstructured DSMs do not take into account the linguistic structure of the text when computing the co-occurrences, this means that a co-occurrence is counted whenever the target term occurs in or close to the context, disregarding the type of linguistic relation between these two elements. On the other hand, structured DSMs extract triples from the corpus, the triples contain word pairs and a syntactic relation or lexico-syntactic pattern between the two words. The context of a target term must be linked to it by some significant lexico-syntactic relation, which is also typically used to distinguish the type of this co-occurrence. The triples are mapped directly onto a two-dimensional matrix, either by dropping one element from the tuple (Pado and Lapata 2007) or by concatenating two elements.

Weighting the contexts: The weight deals with "reducing" the impact of the less important features in a vector in order to improve the information retrieval performance. Computing the similarity between all pairs of vectors in a semantic space is computationally expensive. The similarity must be computed only for those vectors that share non-zero features but most of these shared non-zero features correspond to contexts that are very common and have little semantic discrimination power (e.g., the word "the"). In order to keep only the most informative features some relevance weighting schemes can

be applied to the co-occurrence frequencies, like tf-idf (Spark Jones, 1972) or an statistical association measure like mutual information (Church & Hankes, 1989). Using pointwise mutual information (PMI), the dimensions that co-occur frequently with only very few words, thus correspond to highly discriminating contexts, get a high weight. Using PMI the number of comparisons needed to compare vectors greatly decreases while losing little precision in the similarity (Lin, 1998).

Similarity measure: There are several distance measures that can be used to measure the similarity between two vectors: Euclidean distance, Manhattan, Minkowski p-distance, Kullback-Leibler, cosine similarity. The most popular one is the cosine similarity which measures the cosine of the angle between two vectors. The cosine similarity is calculated as follows:

$$cos(x, y) = \frac{\sum_{i=1}^{n} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \cdot \sqrt{\sum_{i=1}^{n} y_i^2}}$$

$$cos(x, y) = \frac{x \cdot y}{\sqrt{x \cdot x} \cdot \sqrt{y \cdot y}}$$

$$cos(x, y) = \frac{x}{||x||} \cdot \frac{y}{||y||}$$

Where x and y are two vectors, each with n elements. The cosine of the angle between two vectors is the inner product of the vectors after they have been normalized to unit length. If the cosine of the angle is equal to 1 then vectors are collinear, if the value is equal to 0 then the vectors are orthogonal.

Dimensionality reduction: Typically a co-occurrence matrix is very big and sparse, for instance it can be a 1M $\times$ 1M matrix with less than 0.05% cells containing nonzero counts (Evert, 2010). There are several dimensionality reduction techniques that try to compress the matrix. Feature selection is a technique that selects a subset of informative features, this means that only a subset of columns of the co-occurrence matrix is kept. The metrics to decide which features to select include information gain (Mitchell, 1997) and chi-squared test (Liu and Setiono, 1995).

Another popular technique is singular value decomposition (SVD) which is a linear algebra technique for factorizing a matrix (Deerwester,1990). The general idea is to obtain a new reduced matrix that approximates the original co-occurrence matrix and preserves most of the variance. Using SVD, a $n \times m$ matrix can be reduced to a $k \times m$ matrix, where k << n and k < m. These new k dimensions are called latent dimensions, k is an arbitrary choice.

SVD decomposes a matrix X into the product of three matrices $U\Sigma V^T$, where U and V are in column orthonormal form (the columns are orthogonal and have unit length, $U^TU = V^TV = I$) and Σ is a diagonal matrix of singular

		contexts					
		get	see	use	hear	eat	kill
	knife	51	20	84	0	3	0
	cat	52	58	4	4	6	26
target	dog	115	83	10	42	33	17
words	boat	59	39	23	4	0	0
	cup	98	14	6	2	1	0
	pig	12	17	3	2	9	27
	banana	11	2	2	0	18	0

$sim(cat, dog) = cos((52, 58, 4, 4, 6, 26), (115, 83, 10, 42, 33, 17)) = 0.90704$
$sim(banana, dog) = cos((11, 2, 2, 0, 18, 0), (115, 83, 10, 42, 33, 17)) = 0.63196$

Table 2.1: Example, co-occurrence matrix M

values (Golub & Van Loan, 1996). Dimensionality reduction techniques such as SVD have been shown to be effective in many semantic tasks, improving the DSMs performance.

Table 2.1 shows a very simple example of a unstructured DSM, each row vector contains the number of times that the corresponding target word co-occurs in the same sentence with the words in the columns (context). The similarity is calculated by the cosine of the angle between two row vectors.

DSMs are able to detect different kinds of semantic similarity. The kind of similarity measured by the cosine of the angle between row vectors in a word-context matrix is known as *attributional similarity*, which includes standard taxonomic semantic relations such as synonymy, co-hyponymy, and hypernymy. Unstructured DSMs can capture attributional similarity. There is also *relational similarity* which is the property shared by pairs of words linked by similar semantic relations (e.g., hypernymy) despite the fact that the words in one pair might not be attributionally similar to those in the other pair (Baroni and Lenci 2010; Turney 2006). Relational similarity is useful for finding analogies, grouping concept pairs into relation classes, etc. Structured DSMs can capture relational similarity.

This has given rise to a wide range of applications of DSMs in the field of computational linguistics. Some of these applications include: synonym detection, concept categorization, word sense disambiguation, query expansion in information retrieval, ontology & wordnet expansion, probabilistic language

models, textual entailment detection and so on.

DSMs have mostly focused on representing the meaning at the word level, however, modeling the meaning of more complex linguistic constituents is also very important in semantics. In the next section we will talk about compostionality, a key notion for deriving the meaning of a complex expression.

2.2 Compositionality

Formal semantics (FS) treat natural language as a formal language. The semantics are approached by theoretical models where it is possible to define an algorithm to compose the meaning representation of a sentence out of the meaning representation of its single words. The principle of compositionality, also known as Frege's Principle (Frege, 1892), states that the meaning of an utterance is determined by the meanings of its constituents and the rules used to combine them. This principle is used in formal semantics to build up the meaning of a sentence by syntax driven composition of the meaning of its words. The linguistic meanings are represented as symbolic formulas and the meaning of a sentence is its truth value after assembling these linguistic meanings via composition rules.

Linguistic meanings are represented by predicate-argument structures used in logic, e.g., the meaning representation of "Maria eats soup" would be *eat(Maria,soup)*. Frege proposed the first order logic symbols $\exists, \forall$ to represent the meaning of the quantifiers "some" and "all" and avoid ambiguities. For example, the meaning representation of "A person eats soup" would be $\exists x.person(x) \wedge eats(x, soup)$. A sentence is equivalent to a proposition and its meaning is the truth value of its meaning representations, e.g., $[[\exists x.person(x) \wedge sleep(x)]] = 1$. One of the weaknesses of this approach is that, although provides a sophisticated model of sentence meaning, it is usually not enough to deal with the complexity of a natural language.

Distributional approaches to compositionality attempt to derive a distributional, vector-based, representation of a composite phrase from the distributional representations of its parts. One of the possibilities to do this composition is to perform algebraic operations on the semantic vectors representing words. These operations include addition, multiplication, tensor product, dilation (Mitchell and Lapata, 2010). Another approach is to look at corpus-harvested phrase vectors to learn composition functions that should derive such composite vectors automatically. In (Baroni and Zamparelli, 2010) a

new method to derive distributional representations for adjective-noun composition is proposed, nouns are represented as vectors and the adjectives are linear functions from a vector (the noun representation) to another vector (the adejective-noun representation). It is still not clear which is the best way to compose the representation of content words in vector spaces, in any case the long-term goal is to be able to compose larger and larger constituents up to full sentences.

2.3 Entailment

In logic and formal semantics, entailment is a relation that holds between two propositions p and q such that if the truth of q necessarily follows from the truth of p (and the falsity of q necessarily follows from the falsity of p) then p entails q. We can also see it as a relation between sentence meanings: there is an entailment relation between two sentences (or between a set of sentences and a sentence) if the truth of the latter sentence (consequent) necessarily follows from the truth of the former sentence (antecedent), e.g., "there is a dog" entails "there is an animal" since one can not both assert the first and deny the second. Usually, the entailment relation is expressed by the symbol $\models$ (Lyons, 1995). There are a number of other semantic notions closely related to entailment such as logical equivalence, contradiction and logical truth (validity) (Chierchia and Ginet 2001).

So far, we have explained the notion of entailment as a relation between sentences. However, the entailment relations can be defined not only among sentences but also among words (lexical entailment) and phrases (phrase entailment). Each semantic domain A has its own entailment relation $\models_A$. For instance, the entailment relation $\models_S$ corresponds to the entailment relation between sentences explained before. It can also exist an entailment relation between nouns, adjectives, verbs, modifiers, connectives and quantifier phrases. Typically, a sentence denotes a truth value (true or false) or truth conditions, a noun denotes a set of entities, and a quantifier phrase (e.g 'all dogs') denotes a set of sets of entities. So the entailment relation between nouns $\models_N$ are the inclusion relations among sets of entities (hammer $\models$ tool), the entailment between quantifier phrases $\models_{QP}$ are the inclusion relations among sets of sets of entities (all $\models$ many) (Baroni, Bernardi, Do and Shan, 2012; Barwise and Cooper, 1981).

FS school has focused mostly on the behavior of logical words. Logical words are those words that correspond to logical operators like quantifiers, negation,

conjunction, disjunction. A quantifier (e.g. all, many) is a type of determiner that indicates quantity.

A slightly different focus on entailment is represented by textual entailment, which is of high interest in natural language processing (NLP) research. Textual entailment is the task of deciding, given two text fragments, whether the meaning of one text is entailed (can be inferred) from the other text (Dagan and Glickmant and 2004). This focus differs from the traditional logic one, since it allows cases in which the truth of the consequent is highly plausible for most practical purposes, rather than certain, e.g., $\Delta \models \phi$ if typically a human that reads Δ would most likely infer that ϕ is also true. This task captures generically a broad range of inferences that are relevant for multiple applications, for example, question answering (QA) systems.

In distributional semantics, the work on entailment has mostly focused on detecting inferences at the word level (lexical entailment) extracted from distributional vectors representing words, for instance, identifying hyponymy relations from the distributional vectors. Chapter 3 is dedicated to this topic and describes the related work for detecting entailment not only between nouns but between quantifier phrases using distributional semantics. We will see in that chapter that a fundamental component for classifying the relation between a pair of distributional vectors as entailment or non-entailment is the use of machine learning techniques. Next section (2.4) introduces the main notions of machine learning and support vector machines.

2.4 Machine learning

According to Mitchell (2006) the field of Machine Learning seeks to answer the question of how can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes. We say that a machine learns with respect to a particular task T, performance metric P, and type of experience E, if the system reliably improves its performance P at task T, following experience E. In other words, machine learning is about programming computers to optimize a performance criterion using example data or past experience.

Machine Learning represents an intersection of computer science and statistics. While computer science deals with building machines that solve problems, and tries to identify which problems are inherently tractable/intractable, the theory of statistics is needed for building mathematical models which can make

inferences from a sample. Machine learning is needed for the problems in which there is no algorithm or is not easy to write a computer program to solve it, e.g., when human expertise does not exist, the solution changes in time or it needs to be adapted to particular cases. Some of the important aspects that must be taken into account when designing a machine learning system include (Alpaydin, 2004):

Selection of the training experience: As it was mentioned before, a system learns from experience E if its performance at a task T (measured by P) improves with experience E. For instance, if we want to build a spam-filter system, the task T would be to predict which e-mails are spam, the performance metric P could be the percentage of e-mails correctly predicted and a useful training experience E would be a set of e-mails manually labeled as spam and not spam. In this sense, it is important to select a training experience suitable for our learning task, the training experience should be representative of the future situations faced by the learner.

Choosing the target function: The problem of improving performance can often be reduced to the problem of learning some particular target function. A target function $F : X \to Y$ maps all inputs with its correspondent output in a learning task. For instance, in the case of the spam-filter system, the target function would map every mail with its correspondent label (spam or not spam), we want to approximate the target function as good as possible. The target function can be defined in several ways depending on the learning task, e.g., $F : X \to \{0, 1\}$, $F : X \to a\,set\,of\,labels$, $F : X \to R^+$.

Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples (inductive learning hypothesis).

Choosing a representation for the target function: The representation of the target function can be a linear/polynomial function, a set of rules, a neural network, etc. Usually expressive representations can be used for a close function approximation and simple representation for simple training data and learning algorithms. There is a trade-off between the expressiveness of a representation and the ease of learning. The more expressive a representation, the better it will be at approximating an arbitrary function; however, the more examples will be needed to learn an accurate function.

Choosing a learning algorithm: A learning algorithm must be selected in order to approximate or "learn" the target function, for instance, it can be a regression based algorithm, rule induction, genetic algorithm or back-propagation. In the next subsection we mention some of the main types of machine learning.

2.4.1 Types of learning

Supervised learning: This kind of learning needs a set of examples, training data, for finding a description that is shared by all positive examples and none of the negative examples. Doing this, we can make predictions of unseen instances. The input of this inferred function is an instance (typically a vector) and the output should be the desired value, this function is called a classifier if the output is discrete or a regression function if the output is continuous.

Classification is the task of determining to which class or label an unobserved instance belongs to, based on a training data set containing labeled instances. An algorithm that implements classification (find the function that maps input data to a category) is known as a classifier. There are many examples of classification tasks, e.g., the spam-filter system is a classification task since it has to assign one of the two labels "spam" or "not spam" to new e-mails.

In a regression task, the goal is to find the function that predicts a real-valued output given an input instance. The training data consist on instances and their desired output values. The task of finding a inferred function from the training data is an interpolation task. One example of a regression task would be to predict the price of a used car based on several car attributes.

Unsupervised learning: As we mentioned before, in supervised learning the aim is to learn a mapping from the input to an output where correct examples are provided by training data. In unsupervised learning, there is no training data and we only have input data. Here the aim is to find the regularities in the input. One method is clustering where the goal is to find clusters or groupings of the input. Clustering is a very popular technique with a lot of applications, e.g., given a collection of text documents, organize them according to their content similarities to produce a topic hierarchy.

Reinforcement learning: In some applications, the output of the system is a sequence of actions. In such a case, a single action is not important; what is important is the policy that is the sequence of correct actions to reach the goal. In reinforcement learning, the machine learning program should be able to assess the goodness of policies and learn from past good action sequences to be able to generate a policy. This kind of learning is useful in game playing algorithms.

2.4.2 Evaluation methods

The evaluation of a machine learning system performance is conducted experimentally, there are several evaluation methods and metrics. In supervised learning, the available dataset is usually segmented into training set (used to

train the system), validation set (this is optional and it used to optimize the parameters of the system) and test set (used to evaluate the learned system). The instances in the test set cannot be used in any way in the training (learning) of the system. There are several evaluation methods such as: hold-out, stratified sampling, repeated hold-out, cross-validation and bootstrap sampling.

The performance of a learning system can be measured mainly by its predictive accuracy (how accurate the system makes predictions on the test instances). For classification problems a variety of measures has been proposed, this includes precision and recall, tipycally used in information retrieval. Other performance evaluation metrics take into account efficiency (time and memory), robustness, scalability, interpretability and complexity. We will explain two types of evaluation that are common and that we used for the evaluation of our experiments.

In hold-out evaluation, the entire dataset is divided into two disjoint subsets, the training set and the test set. Usually the training set is a much more bigger than the test set. Every instance included in the test set should not be used in the training phase. The unseen test instances in the test set provide an unbiased estimate of the systems predictive accuracy.

In a k-fold cross-validation, the entire dataset is partitioned into k disjoint subsets, called folds, of approximately equal size. Each fold in turn is used as the test set and the remainder (i.e., (k-1) folds) as the training set. The k error rates are averaged to produce the overall error estimate. The value of k is arbitrary, a common choice is 10 or 5. The advantage of this kind of evaluation is that all the data can be used for training, it is suitable when the entire dataset is not large.

2.4.3 Support vector machines

This section provides an overview of the theoretical foundations of the support vector machines (SVMs) and the effect of the parameters involved in this learning method, we included this section because the decision of which types of SVM classifiers to use in the experiments (Chapter 4) was based on these notions. SVMs (Boser, Guyon, and Vapnik 1992) are a supervised machine learning method for classification and regression problems, where the general idea is to find a hyperplane that separates two classes of data. While some learning methods find just any linear separator, SVM looks for a separator (hyperplane) that is maximally far away from any data point. The distance from this hyperplane to the closest data point determines the margin of the

classifier. This method of construction necessarily means that the decision function is fully specified by a (usually small) subset of the data which defines the position of the separator. These points are referred to as the support vectors, other data points play no part in determining the decision surface that is chosen (Manning, Raghavan, Schütze, 2008).

When the data is not linearly separable (there is no linear model/hyperplane that separate the two classes) SVMs are able to map the data to a higher dimensional space and find a separating hyperplane in the new space. The linear model in the new space (feature space) corresponds to a nonlinear model in the original space (input space), the transformation is done by using kernel functions and this is the reason why SVMs are sometimes generalized under the name "kernel machines"(Alpaydin, 2004). Figure 2.1 shows a graphical example.

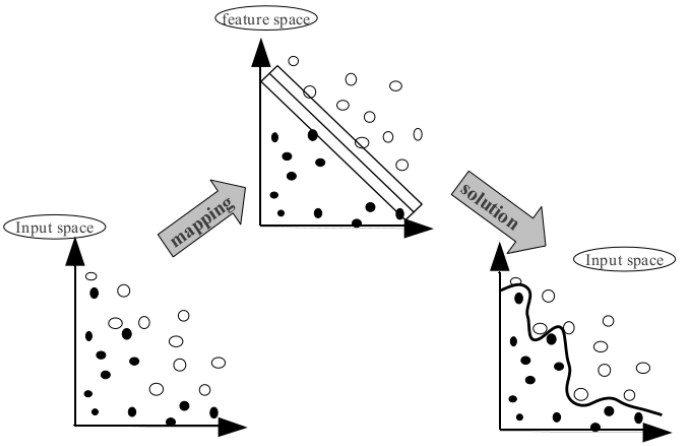


Figure 2.1: SVM, a two-dimensional example

The linearly separable case.

For two-class separable training data sets, there are lots of possible linear separators. SVM, in particular, looks for a decision surface that is maximally far away from any data point. Maximizing the margin allows better generalization and points near the decision surface represent very uncertain classification decisions: there is almost a 50% chance of the classifier deciding either way (Manning, Raghavan, Schütze, 2008). A classifier with a large margin makes no low certainty classification decisions, it is proven that maximizing the margin minimizes the upper bound of the classification error.

Formally, the two data classes are labeled with -1/+1 and the set of training data points is expressed by $D = \{[x_i, y_i], i = 1, ..., l, y_i \in \{-1, 1\}, x_i \in R^d\}$ where each member is a pair of a point and a class label corresponding to it, when we say data point we are actually refering to vectors of the form $x_i = (x_1, x_2, ..., x_d)$. If we want to find a linear separator of the two classes, we must find w and b such that:

$$x_i \cdot w + b \geq 1 \text{ for } y_i = +1 \tag{2.1}$$

$$x_i \cdot w + b \leq -1 \text{ for } y_i = -1 \tag{2.2}$$

This can be combined into one set of inequalities:

$$y_i(x_i \cdot w + b) - 1 \geq 0 \ \forall i \tag{2.3}$$

Commonly, in the machine learning literature, w is known as the weight vector and b as the bias. The decision hyperplane (the hyperplane that separates positive from negative examples) is defined by b and a normal vector w which is perpendicular to the hyperplane. The equation of the decision hyperplane is $w \cdot x + b = 0$, the points x that satisfy this equation lie on the hyperplane, w is normal to the hyperplane, $\frac{|b|}{\|w\|}$ is the perpendicular distance from the hyperplane to the origin, and $\|w\|$ is the euclidean norm of w.

However, SVM classifiers are not only interested in finding a hyperplane that discriminates correctly the training data points (there are many possible hyperplanes), they try to find the one with the maximum distance from the hyperplane to the instances closest to it on either side; in other words, the goal is to maximize the margin around the decision boundary. Let d_+, d_- be the shortest distance from the separating hyperplane to the closest positive (negative) instance, then the margin of the separating hyperplane is defined by $\rho = d_+ + d_-$. If we consider the points for which the equality in (2.1) holds, these points lie in the hyperplane $H_1 : x_i \cdot w + b = 1$ with normal w and perpendicular distance from the origin $\frac{|1-b|}{\|w\|}$. Similarlly, the points for which the equality in (2.2) holds lie on the hyperplane $H_2 : x_i \cdot w + b = -1$ with normal w and a perpendicular distance from the origin $\frac{|-1-b|}{\|w\|}$. Hence $d_+ = d_- = \frac{1}{\|w\|}$ and the margin is $\rho = d_+ + d_- = \frac{2}{\|w\|}$. H_1 and H_2 are parallel, they have the same normal, and no training points fall between them, thus the pair of hyperplanes which gives the maximum margin is found by minimizing $\|w\|^2$ subject to constraints in (2.3). Those training data points for which the equality in (2.3) holds (the points that lie in H_1 or H_2) and whose removal would change

the solution found, are called support vectors, fig. 2.2 (Burges 98).

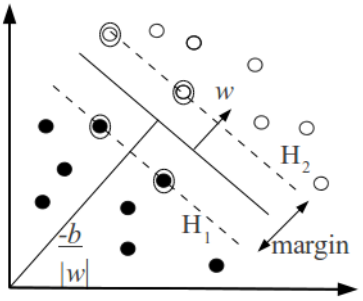


Figure 2.2: The decision boundary for the separable case. The support vectors are circled

In fact, learning in SVM can be formulated as an optimization problem:

$$min \|w\|^2 \text{ subject to } y_i(x_i \cdot w + b) \geq 1 \forall i \tag{2.4}$$

Or equivalently,

$$max \frac{2}{\|w\|} \text{ subject to } \begin{cases} x_i \cdot w + b \geq 1 \text{ for } y_i = +1 \\ x_i \cdot w + b \leq -1 \text{ for } y_i = -1 \end{cases} \tag{2.5}$$

This is a standard quadratic optimization problem subject to linear constraints. Quadratic optimization problems are a well-known class of mathematical optimization problems, many algorithms exist for solving them. For the case of the SVM optimization problem, the solution involves constructing a dual problem where a Lagrange multiplier α_i is associated with each constraint in the primal problem (2.3). This method is useful because replacing the constrains in (2.3) by constraints on the Lagrange multipliers makes them easier to handle. Also because with this reformulation of the problem, the training data will only be expressed in the form of dot products between vectors, this is a crucial property that allows SVM to generalize the procedure to the nonlinear case as we will see further. The dual optimization problem can be expressed as:

Find $\alpha_1, ..., \alpha_n$ such that $\sum \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j$ is maximized, and:
- $\sum_i \alpha_i y_i = 0$
- $\alpha_i \geq 0$

The solution is then of the form:
$w = \sum_i \alpha_i y_i x_i$
$b = y_k - wx_k$ for any x_k such that $\alpha_k \neq 0$

In the solution, most of the α_i are zero. Each non-zero α_i indicates that the corresponding xi is a support vector. The classification or decision function of an SVM is then:

$$f(x) = sign(\sum_i \alpha_i y_i x_i x + b) \tag{2.6}$$

The classification function involve a dot product between pairs of vectors (x and x_i).

The linearly nonseparable case

If the training set D is not linearly separable, one approach is to allow the decision margin to make a few mistakes. We then pay a cost for each misclassified example, which depends on how far it is from meeting the margin requirement given in Eq.(2.3). This approach is usually known as soft margin classification (Cortes and Vapnik, 1995). To implement this, slack variables are introduced, $\xi_i, i = 1, ..., l$ which store the deviation from the margin. There are two types of deviation: An instance may lie on the wrong side of the hyperplane and be misclassified. Or, it may be on the right side but lie inside the margin, not sufficiently away from the decision hyperplane. If $\xi_i = 0$ it means that x is correctly classified. If $0 < \xi_i < 1$, it means x is correctly classified but lies on the margin, If $\xi_i \geq 1$, it means x is misclassified. A non-zero value for ξ_i allows x_i to not meet the margin requirement at a cost proportional to the value of ξ_i.
The SVM optimization problem introducing slack variables can be expressed as a dual optimization problem of the form (we have skipped the detailed steps):

Find $\alpha_1, ..., \alpha_n$ such that $\sum \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j$ is maximized, and:
- $\sum_i \alpha_i y_i = 0$
- $0 \leq \alpha_i \leq C$

The difference with the optimal linearly separable case is that now α_i have an upper bound of C. This optimization problem involves a trading off between how fat the margin can be versus how many points have to be moved around to allow this margin. The soft margin constant C is a parameter chosen by the user, a larger C corresponds to assigning a higher penalty to errors. The parameter C plays the role of a regularization term which provides a way to

control overfitting: as C becomes large, it is "unattractive" to not respect the data at the cost of reducing the geometric margin; when it is small, it is easy to account for some data points with the use of slack variables and to have a fat margin placed so it models the bulk of the data (Manning, Raghavan, Schütze, 2008).

The solution of the dual problem is of the form:
$w = \sum_i \alpha_i y_i x_i$
$b = y_k(1 - \xi_k) - wx_k$ for $k = \text{argmax}_k \alpha_k$

As before, the x_i with non-zero α_i will be the support vectors. Typically, the support vectors are a small proportion of the training data. However, if the problem is non-separable or with small margin, then every data point which is misclassified or within the margin will have a non-zero α_i. If this set of points becomes large, then, this can be a slowdown for using SVMs at test time.

There is another alternative when the dataset does not allow a classification by a linear classifier: nonlinear SVMs. In reality, nonlinear SVMs instead of trying to fit a nonlinear model, map the data on to a higher dimensional space and then use a linear classifier in the higher dimensional space where the training set is linearly separable. In order to accomplish this, kernel functions are used to map the data from the input space X to a feature space F using a non-linear function ϕ: $X \to F$. We have seen before that SVM linear classifiers rely on a dot product between the data point vectors, however, if we decide to map every data point into a higher dimensional space via the transformation ϕ, the dot products become $\phi(x_i) \cdot \phi(x_j)$. A kernel function K is a function that corresponds to a dot product in some expanded feature space, using a kernel function we can easily compute $\phi(x_i) \cdot \phi(x_j)$ (which is just a real number) in terms of the original data points with no need to map or explicitly know ϕ. This is commonly known as the "kernel trick", if we replace $x_i \cdot x_j$ by $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ everywhere in the training algorithm we will produce an SVM classifier that lives in a high dimensional space. All the considerations of the previous sections hold since we are still doing a linear separation but in a different space.

The nonlinear svm classification function is:

$$f(x) = sign(\sum_i \alpha_i y_i \phi(x_i) \cdot \phi(x) + b) = sign(\sum_i \alpha_i y_i K(x_i, x) + b) \qquad (2.7)$$

A kernel function k must be continuous, symmetric and satisfy Mercer's condition (Vapnik, 1995). The most popular choices of families of kernels include

the polynomial kernels and radial basis functions. Polynomial kernels are of the form $K(x, y) = (x \cdot y)^d$ (homogeneous) and $K(x, y) = (x \cdot y + 1)^d$ (inhomogeneous) where d is the degree. A polynomial kernel with $d = 1$ is known as a linear kernel, which corresponds to the linearly separable case explained in the previous sections. A polynomial kernel with d=2 gives a quadratic kernel, and it is very commonly used.

For example, in the case of a quadratic (inhomogeneous) kernel, $K(x, y) = (x \cdot y + 1)^2$ for 2-dimensional vectors $x = (x_1, x_2)$, $y = (y_1, y_2)$. We know K is a kernel function, since there is some ϕ such that $K(x, y) = \phi(x) \cdot \phi(y)$.

Consider $\phi(x) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2]$ (Cherkassky and Mulier 1998), then:

$$\begin{aligned} K(x, y) &= (x \cdot y + 1)^2 \\ &= (x_1y_1 + x_2y_2 + 1)^2 \\ &= 1 + 2x_1y_1 + 2x_2y_2 + 2x_1x_2y_1y_2 + x_1^2y_1^2 + x_2^2y_2^2 \\ &= (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2) \cdot (1, \sqrt{2}y_1, \sqrt{2}y_2, \sqrt{2}y_1y_2, y_1^2, y_2^2) \\ &= \phi(x) \cdot \phi(y) \end{aligned}$$

This shows that the quadratic kernel $K(x, y) = (x \cdot y + 1)^2$ is a dot product in a transformed feature space.

Effect of SVM parameters:

When training an SVM classifier there are certain parameters chosen by the user that may have a significant effect on the decision boundary and therefore in the performance of the classifier. These parameters are typically the soft margin constant C and any parameters the kernel function may depend on, e.g., the degree in the case of the polynomial kernels.

A large value of C assigns a large penalty to errors (Cortes and Vapnik, 1995). Figure 2.3 shows how the value of C can affect the orientation of the hyperplane, a smaller value of C allows to ignore points close to the boundary and increases the margin. For selecting an optimal value, cross-validation can be performed trying several values of C. The selected C will be the one that achieves the best cross-validation accuracy.

The degree of the polynomial kernel controls the flexibility of the decision boundary. The linear kernel, which has the lowest degree, is not sufficient when a non-linear relationship between features exists. Figure 2.4 shows an example of different decision boundaries according to the degree of the kernel. The selection of the optimal kernel is often data-dependent and several

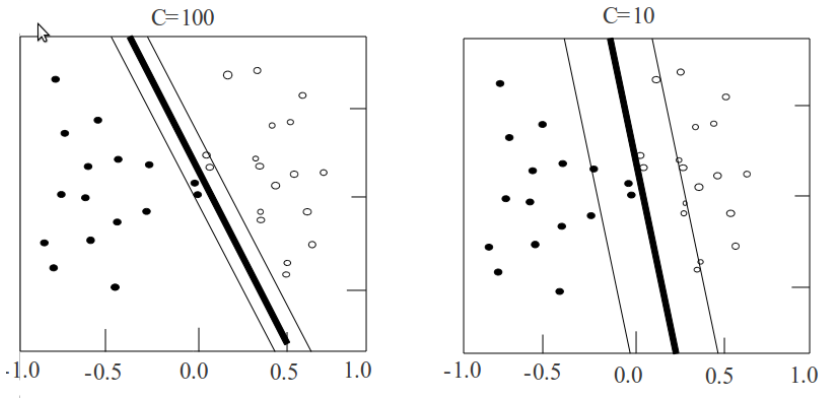


Figure 2.3: Example of the effect of the constant C in the decision boundary.

degrees should be tried. A linear kernel can be tried first and then check if the performance can be improved by using a non-linear kernel. The linear kernel provides a useful baseline and for some applications provides the best results.

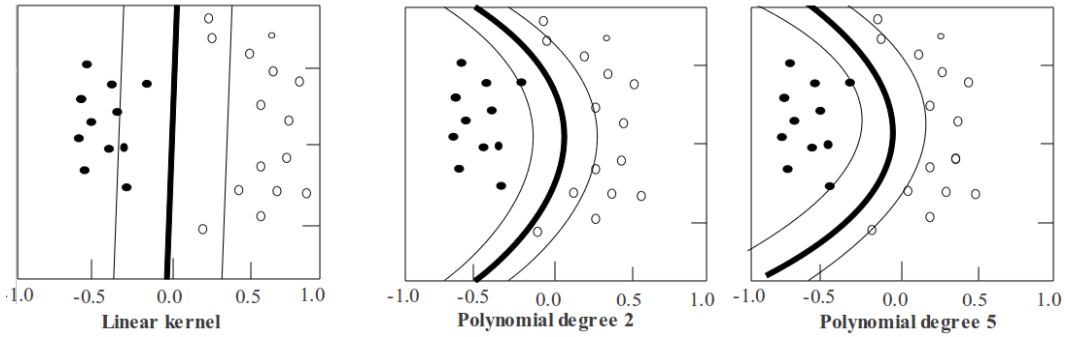


Figure 2.4: Example of the effect of the degree of a polynomial kernel

Chapter 3

Entailment in Distributional Semantics

People working on entailment in distributional semantics have focused on detecting inferences at the word level extracted from distributional vectors representing words. The cosine similarity, which is often used as a similarity measure between row vectors in a DSM, has shown to be a good measure for capturing synonymy, but it seems that it does not capture the "is-a" relation, e.g., "apple is-a food". This is related with the fact that distributional word similarity is most commonly perceived as a symmetric relation, while lexical entailment is in general a directional relation (asymmetric). A way to exemplify these types of relations is imaging an IR system where a user looks for "baby food", he will be satisfied with documents about "baby pap" or "baby juice" ("pap is-a food", "juice is-a food"); but when looking for "frozen juice" he will not be satisfied by "frozen food" since not all foods are juice, even though these two words are semantically related (Kotlerman et al., 2010). Based on this, a proposed notion of lexical entailment is that given a pair of words, the entailment relation holds if there are some contexts in which one of the words can be substituted by the other, such that the meaning of the original word can be inferred from the new one. This corresponds to an asymmetric relation, for example, baseball contexts are also sport contexts but not vice versa, hence baseball is 'narrower' than sport and baseball $\models$ sport (Kotlerman et al., 2010).

Few works have investigated asymmetric similarity measures (Weeds and Weir, 2003; Geffet and Dagan, 2005; Bhagat et al., 2007; Szpektor and Dagan, 2008). Kotlerman et al. (2010) proposed one asymmetric measure to capture the entailment relation between distributional vectors, based on the notion of feature inclusion. This measure, called balAPinc, is based on the common expectation that the context features characterizing an entailing word should be largely in-

cluded in those of the entailed word. The intuition behind the measure is that: the relation "is-a" scores higher if included features are ranked high for the narrow term, "is-a" scores higher if included features are ranked high in the broader term vector as well, "is-a" scores lower for short feature vectors. The measure balAPinc measure is defined as:

$$balAPinc(u \models v) = \sqrt{APinc(u \models v) \cdot LIN(u, v)} \tag{3.1}$$

balAPinc captures a relation of feature inclusion between the narrower (entailing) and broader (entailed) terms. It is based on Information Retrieval methods to evaluate different ranking systems but addapted to lexical inclusion. balAPinc is a geometric average between two terms, APinc and LIN (Lin, 1998):

$$APinc(u \models v) = \frac{\sum_{r=1}^{|F_u|}[P(r) \cdot rel'(f_r)]}{|F_u|} \tag{3.2}$$

$$LIN(u, v) = \frac{\sum_{f \in F_u \cap F_v}[w_u(f) + w_v(f)]}{\sum_{f \in F_u} w_u(f) + \sum_{f \in F_v} w_v(f)} \tag{3.3}$$

APinc (eq. 3.2) is a version of the Average Precision measure used in information retrieval (Voorhees and Harman, 1999). F_u and F_v represent the features, with positive PMI values, of the semantic vectors of the candidate pair $u \models v$. The idea is that, if u and v are in an entailment relation then the features that have larger values in F_u should also have have large values in F_v. The F_u features are ranked according to their PMI value so that f_r is the feature in F_u with rank r, $P(r)$ is the precision at r which is higher when highly ranked u features are present in F_v as well, $rel'(f_r)$ is the relevance term which is higher when the feature f_r in F_u also appears in F_v with a high rank. On the other hand, LIN (eq. 3.3) measures feature vector overlap and it is used to balance the potentially excessive asymmetry of APinc towards the features of the antecedent. $w_u(f)$, $w_v(f)$ are the PMI values of the feature f in the vectors F_u and F_v.

The entailment relation explained so far was focused on nouns, the question if entailment can be detected using distributional semantic representations of phrases and if it generalizes to phrases containing logical words like quantifiers has been raised by Baroni, Bernardi, Do and Shan (2012). They focus on entailment among composite phrases rather than nouns and in entailment among logical words rather than content words, suggesting that there are different entailment relations at different semantic types. This work is of special

interest for this thesis since we aim to repeat and enhance the experiments that involved an SVM classifier; the detailed description of the original experiments and the obtained results are explained in the subsections 3.1 and 3.2 of this chapter.

The main idea presented in Baroni, Bernardi, Do and Shan (2012) is to characterize entailment between nouns by collecting semantic vectors exemplifying which noun entails which and then to apply a supervised machine learning method to predict if there is an entailment relation between unseen pairs of nouns. They use a cheap way to collect such training data with almost no manual effort: they extract adjective-noun sequences (AN) and build AN $\models$ N patterns as positive examples of entailment, based on the notion that most ANs entail their head Ns. The results show that training the classifier with AN $\models$ N (big cat $\models$ cat) examples, allows to predict the entailment relation between unseen pair of nouns N $\models$ N (dog $\models$ animal).

They also study the entailment between quantifier-noun sequences (QN) such that Q1N $\models$ Q2N . They study 12 quantifiers and identified 13 clear cases where Q1 $\models$ Q2 and 17 clear cases where Q1 $\not\models$ Q2. The goal is to train the classifier with QNs that are in an entailment relation (many dogs $\models$ some dogs) and be able to predict entailment between unseen quantifiers (all cats $\models$ several cats).

In the next two subchapters we are going to describe in the experiments conducted in Baroni, Bernardi, Do and Shan (2012) for detecting entailment using distributional semantic representations. It is important to explain the configuration of these experiments, specially the one for detecting entailment between quantifiers, since they were repeated for the aim of this thesis with modifications on the machine learning classifier. Our modifications are discussed in Chapter 4.

3.1 Detecting noun entailment

In chapter 2, it was mentioned that one of the first steps in a DSM is to build a semantic space or co-occurrence matrix that captures the distribution of the words or phrases of interest accross contexts. In the entailment detection experiments performed in Baroni, Bernardi, Do and Shan (2012), the corpora used to extract the distributional semantics vectors were the British National Corpus (http://www.natcorp.ox.ac.uk/), WackyPedia and ukWaC (http://wacky.sslmit.unibo.it) with a total amount of 2.83-billion-tokens. The

corpora were tokenized, POS-tagged and lemmatized to merge singular and plural instances of words and phrases. The phrases of interest (rows of the matrix) include: AN (*big cat*) and QN sequences (*all cats*), the adjectives (*big*), quantifiers (*all*) and nouns (*cat*) contained in those sequences, and the most frequent nouns and adjectives in the corpora. The contexts (columns) included the most frequent 9.8K nouns, 8.1K adjectives, and 9.6K verbs in the corpora. For building the distributional semantic vectors, co-occurrences were counted between the phrases of interest and the content words appearing in the same sentence. These co-occurrence counts were converted to PMI. The result is a co-occurrence matrix with 48K rows (one per phrase of interest) and 27K columns (one per content word).

Additionally, the co-occurrence matrix was transformed using SVD in order to reduce the dimensionality to only 300 columns. These reduced vectors were used for feeding the SVM classifiers.

For training a classifier that is able to predict which noun entails which, training data that exemplify this relation is needed. Baroni, Bernardi, Do and Shan (2012) introduce a cheap way to collect such training data by extracting adjective-noun sequences (AN) and building AN $\models$ N patterns as positive examples of entailment, based on the notion that most ANs entail their head Ns (e.g., big cat $\models$ cat). From the distributional point of view the vector representing an AN should include the information encoded in the vector representing N, since AN occurs at least in the same contexts than N.

The positive examples of this training data, called the AN$\models$N dataset, were created by simply concatenating the semantic vectors of ANs and Ns (e.g., big cat $\models$ cat). The negative examples were created by randomly permuting the Ns, since an AN usually does not entail another N (e.g., big cat $\not\models$ dog). Most of the frequent adjectives are not restrictive, in order to reduce the noise caused by these adjectives and to assure that the phenomenon of entailment is being examined, 256 restrictive adjectives were manually selected from the most frequent 300 adjectives in the corpus. Then it was performed the Cartesian product between these adjectives with 200 concrete nouns extracted from BLESS data set (Baroni and Lenci, 2011) to avoid highly polysemous words. Finally, after the Cartesian product, a total of 1246 AN sequences were obtained. The AN$\models$N dataset contains those 1246 positive instances of AN$\models$N entailment and an equal amount of negative examples $AN_1 \not\models N_2$ created by randomly permuting the nouns in the positive instances.

The goal is to train a classifier with the AN$\models$N dataset and use it to predict lexical entailment of the type $N_1 \models N_2$ e.g., training the classifier with pairs such as big cat$\models$cat and test it on pairs such as dog$\models$animal. In order to obtain pairs useful for testing data, all Wordnet nouns in the corpus were listed and the hyponym-hypernym chains were extracted. For example, *cat* is found to entail *carnivore* because WordNet contains the chain cat$\rightarrow$feline, feline$\rightarrow$carnivore. From here, 1385 positive instances of $N_1 \models N_2$ entailment were extracted and the same amount of negative instances $N_1 \not\models N_2$ were created by inverting and by randomly shuffling the words across the positive instances. In all the cases it was manually checked that the positive and negative instances were in an entailment and non-entailment relation respectively. This testing dataset was called the $N_1 \models N_2$ dataset.

An SVM classifier was used to predict the lexical entailment. The semantic vectors are the concatenation of two vectors, for instance, the testing instances are the concatenation of two vectors representing two nouns and the output is $\{-1, 1\}$ where 1 means that there is an entailment relation between the two nouns and -1 indicates the opposite case. In all the cases the dimensionality of the semantic vectors is 600 (300 per each member of the entailment pair) since SVD was applied to reduce the dimensions of the original semantic space to 300.

The classifier was evaluated by classifying the pairs in the $N_1 \models N_2$ dataset, trained on the AN$\models$N dataset ($\text{SVM}_{AN\models N}$). Another evaluation regime was to classify the pairs in the $N_1 \models N_2$ dataset, training the classifier with 10-fold cross-validation on the $N_1 \models N_2$ dataset itself (SVM_{upper}).

Additionally, the balAPinc measure (3.1) that captures the relation of feature inclusion, was used to recognize entailment. A threshold t was selected above which a pair was classified as entailing or not. The threshold t was determined in two different ways. In balAPinc_{upper} the selected threshold t is the one that maximizes the F-measure on the test set ($N_1 \models N_2$ dataset) giving an upper bound on how well balAPinc could perform on the test set. In $\text{balAPinc}_{AN\models N}$ the selected threshold t is the one that maximizes the F-measure on the AN$\models$N data set. The balAPinc measure was used as a reference point for comparing the performance obtained using SVM classifiers, since balAPinc represents the state of the art in various tasks related to lexical entailment.

Two baseline methods were designed to compare if the performance of the entailment recognizers was above chance. The first baseline $\text{fq}(N_1)<\text{fq}(N_2)$

	P	R	F	Accuracy(95% C.I.)
SVM_{upper}	88.6	88.6	88.5	88.6 (87.3–89.7)
$\text{balAPinc}_{AN\models N}$	65.2	87.5	74.7	70.4 (68.7–72.1)
balAPinc_{upper}	64.4	90.0	75.1	70.1 (68.4–71.8)
$\text{SVM}_{AN\models N}$	69.3	69.3	69.3	69.3 (67.6–71.0)
$\cos(N_1,N_2)$	57.7	57.6	57.5	57.6 (55.8–59.5)
$\text{fq}(N_1)<\text{fq}(N_2)$	52.1	52.1	51.8	53.3 (51.4–55.2)

Table 3.1: Detecting lexical entailment.

guesses entailment if the first word is less frequent than the second. The second, $\cos(N_1, N_2)$, applies a threshold (determined on the test set) to the cosine similarity of the pair.

The entailment detection accuracies obtained with the different methods are shown in table 3.1. In all the cases the SVM classifier performs at least as good as the balAPinc measure and considerably better than the baseline classifiers. The results suggest that the distributional vectors of the AN$\models$N dataset are excellent training data for discovering entailment between nouns.

3.2 Detecting QN entailment

The semantic space used for this experiment is the same described in the previous section. As in the previous experiments, SVM classifiers were used

For detecting entailment among quantifier phrases, the classifier must learn from training data containing pairs of the form $Q_1N\models Q_2N$. This training dataset, called the $Q_1N\models Q_2N$ dataset, was built by performing the Cartesian product between a fixed set of quantifiers (*all, both, each, either, every, few, many, most, much, no, several, some*) and 6402 nouns extracted from Wordnet (the same nouns used for the $N_1 \models N_2$ dataset explained in the section before). From this product, a total of 28926 QN sequences, that occur at

least 100 times in the corpus, were obtained.

From the fixed set of quantifiers, 13 clear cases were manually identified where $Q_1 \models Q_2$ and 17 clear cases where $Q_1 \not\models Q_2$. For each of these 30 quantifier pairs (Q_1, Q_2), a set of Wordnet nouns were enumerated such that the semantic vectors of Q_1N and Q_2N were available in the semantic space as phrases of interest (rows). These nouns allowed to build the positive and negative instances of the training dataset since $Q_1N \models Q_2N$ if $Q_1 \models Q_2$ and $Q_1N \not\models Q_2N$ if $Q_1 \not\models Q_2$, e.g., *many dogs* $\models$ *some dogs* since *many* $\models$ *some* and *many dogs* $\not\models$ *most dogs* since *many* $\not\models$ *most*. The set of entailing and non-entailing quantifier pairs are shown in table 3.2.

As in the previous experiment, an SVM classifier was used to predict the entailment between quantifier phrases. The classifiers had a default configuration using Weka 3 [1] and LIBSVM 2.8 (Chang and Lin, 2011), i.e., a cubic polynomial kernel. The input of the classifier is the concatenation of the two vectors representing Q_1N and Q_2N and the output is $\{-1, 1\}$ where 1 means that there is an entailment relation between the two quantifier phrases and -1 indicates the opposite case. The dimensionality of the input vector is 600 (300 per each quantifier phrase).

The aim of this experiment was to train the classifier with entailment pairs of the form $Q_1N \models Q_2N$ (e.g., many dogs$\models$some dogs) and to be able to predict correctly the entailment between pairs $Q_3N \models Q_4N$ (e.g., all cats$\models$several cats) even if Q_3 or Q_4 was not seen at all in the training dataset. Several training and testing regimes were designed in order to see if the information encoded in the QP vectors is enough to generalize the QN entailment:

SVM pair-out. In this regime, one quantifier pair (table 3.2) is held out as testing data and the remaining 29 pairs are used as training data. This means that the classifier must be able to predict correctly the entailment relation $Q_1N \models Q_2N$ without seeing these pair of quantifiers (Q_1,Q_2) in an entailment relation in the training data. For example, the classifier must discover *all dogs* $\models$ *some dog*s without seeing any *all N* $\models$ *some N* examples in the training data. The results and number of instances are shown in table 3.2.

SVM quantifier-out. In this regime one of the 12 quantifiers is held out as testing data, this means that all the pairs containing this quantifier (whether in the antecedent or consequent position) are the test data, the remaining pairs are used as training data. This regime is expected to be more challenging since

[1] http://weka.wikispaces.com/LibSVM

there is less training data and the classifier must predict the entailment without having seen at all one of the quantifiers involved in the entailment pair. For example, the classifier must guess *all dogs* $\models$ *some dogs* without ever seeing *all* in the training data. The results of this regime and number of instances are shown in table 3.3.

SVM$^{Q}_{pair-out}$**, SVM**$^{Q}_{quantifier-out}$**.** These training regimes are the same than the two described before but in this approach the nouns are ignored and only the vectors of the quantifiers are used for training the classifier. For example, the prediction of *all dogs* $\models$ *some dogs* considers only the semantic vectors of the quantifiers *all* and *some*

SVM$_{AN\models N}$**.** In this regime, the classifier must predict the entailment between quantifier phrases but trained on the on the AN$\models$N dataset. This evaluation regime was designed to know if noun-level entailment generalizes to quantifier phrase entailment.

Additionally, the balAPinc measure and two baseline classifiers, cos(QN_1, QN_2) and fq(QN_1)<fq(QN_2) were used to detect entailment between quantifier phrases. Table 3.4 summarizes the performance obtained with all the regimes previously described. SVMpair-out, and $SVM_{\text{quantifier-out}}$ obtained the best performances, suggesting that semantic vectors of QNs carry enough information for predicting entailment. Even SVM$^{Q}_{pair-out}$ and SVM$^{Q}_{quantifier-out}$ performed better than most of the regimes which indicates that even Q vectors alone encode enough information to capture entailment above chance (above the baseline classifiers). The classification using the regime SVM$_{AN\models N}$ obtained a low performance, it performed even worst than one of the baseline classifiers, this result is in accordance with the notion of FS that each semantic domain has its own entailment relation thus noun-level entailment does not generalize to quantifier phrase entailment. Finally, the failure of balAPinc measure in this task suggests that the notion of feature inclusion seems enough to capture entailment between nouns but is not enough for the case of quantifier phrases. A more flexible classifier, like SVM, is more suitable since relies in properties of the vectors beyond feature inclusion.

Quantifier pair	Instances	Correct
all $\models$ some	1054	1044 (99%)
all $\models$ several	557	550 (99%)
each $\models$ some	656	647 (99%)
all $\models$ many	873	772 (88%)
much $\models$ some	248	217 (88%)
every $\models$ many	460	400 (87%)
many $\models$ some	951	822 (86%)
all $\models$ most	465	393 (85%)
several $\models$ some	580	439 (76%)
both $\models$ some	573	322 (56%)
many $\models$ several	594	113 (19%)
most $\models$ many	463	84 (18%)
both $\models$ either	63	1 (2%)
Subtotal	*7537*	*5804 (77%)*
some $\not\models$ every	484	481 (99%)
several $\not\models$ all	557	553 (99%)
several $\not\models$ every	378	375 (99%)
some $\not\models$ all	1054	1043 (99%)
many $\not\models$ every	460	452 (98%)
some $\not\models$ each	656	640 (98%)
few $\not\models$ all	157	153 (97%)
many $\not\models$ all	873	843 (97%)
both $\not\models$ most	369	347 (94%)
several $\not\models$ few	143	134 (94%)
both $\not\models$ many	541	397 (73%)
many $\not\models$ most	463	300 (65%)
either $\not\models$ both	63	39 (62%)
many $\not\models$ no	714	369 (52%)
some $\not\models$ many	951	468 (49%)
few $\not\models$ many	161	33 (20%)
both $\not\models$ several	431	63 (15%)
Subtotal	*8455*	*6690 (79%)*
Total	*15992*	*12494 (78%)*

Table 3.2: Entailing and non-entailing quantifier pairs and $\text{SVM}_{\text{pair-out}}$ (cubic kernel)performance breakdown with number of instances per each case.

Quantifier	Instances		Correct		
	$\models$	$\not\models$	$\models$	$\not\models$	
each	656	656	649	637	(98%)
every	460	1322	402	1293	(95%)
much	248	0	216	0	(87%)
all	2949	2641	2011	2494	(81%)
several	1731	1509	1302	1267	(79%)
many	3341	4163	2349	3443	(77%)
few	0	461	0	311	(67%)
most	928	832	549	511	(60%)
some	4062	3145	1780	2190	(55%)
no	0	714	0	380	(53%)
both	636	1404	589	303	(44%)
either	63	63	2	41	(34%)
Total	*15074*	*16910*	*9849*	*12870*	*(71%)*

Table 3.3: Results of $\text{SVM}_{\text{quantifier-out}}$ training regime (cubic kernel).

	P	R	F	Accuracy (95% C.I.)
$\text{SVM}_{\text{pair-out}}$	76.7	77.0	76.8	78.1 (77.5–78.8)
$\text{SVM}_{\text{quantifier-out}}$	70.1	65.3	68.0	71.0 (70.3–71.7)
$\text{SVM}^{\text{Q}}_{\text{pair-out}}$	67.9	69.8	68.9	70.2 (69.5–70.9)
$\text{SVM}^{\text{Q}}_{\text{quantifier-out}}$	53.3	52.9	53.1	56.0 (55.2–56.8)
$\cos(\text{QN}_1, \text{QN}_2)$	52.9	52.3	52.3	53.1 (52.3–53.9)
$\text{balAPinc}_{AN\models N}$	46.7	5.6	10.0	52.5 (51.7–53.3)
$\text{SVM}_{AN\models N}$	2.8	42.9	5.2	52.4 (51.7–53.2)
$\text{fq}(\text{QN}_1)<\text{fq}(\text{QN}_2)$	51.0	47.4	49.1	50.2 (49.4–51.0)
$\text{balAPinc}_{\text{upper}}$	47.1	100	64.1	47.2 (46.4–47.9)

Table 3.4: Results of quantifier entailment detection.

Chapter 4

Trying different SVM classifiers

In the previous chapter we described the experiments conducted to detect entailment using SVM classifiers and distributional semantic vectors. Baroni, Bernardi, Do and Shan (2012) showed that SVM classifiers are able to capture entailment relations and that the learning model that successfully detects entailment between nouns does not generalize well for detecting entailment between quantifiers since they are different entailment relations. The original experiments payed little attention to the parameters configuration of the classifier, we will repeat the QN entailment experiments using different configurations, starting from the simplest type of classifier (linear) and then with two more classifiers with a polynomial kernel of degree 2; we will perform an optimization procedure to choose an appropriate value of the constant C (soft margin parameter) per each quantifier/quantifer-pair (training regimes $SVM_{pair\text{-}out}$, $SVM_{quantifier\text{-}out}$). Additionally, to reinforce the evidence that there are different entailment relations at different semantic types we will train a classifier on the $Q_1N \models Q_2N$ dataset and test it on the datasets $N_1 \models N_2$ and $AN \models N$, expecting to obtain a low predicting accuracy.

We saw that, for the task of QN entailment, the SVM classifier performed better than the balAPinc classifier which only takes into account individual features of one vector that are included in the other. Our hypothesis of the successful behavior of SVM in this task is that SVM classifiers, using a polynomial kernel of degree greater than one, take into account not only the influence of individual features but the combination between them, e.g., the product of the first dimension of the antecedent QN and the first dimension of the consequent QN. In the work of Baroni, Bernardi, Do and Shan (2012) the question of what properties of the vectors are taken into account for predicting an entailment is left as future work. Repeating the experiments with lower degree of polynomial kernel is also a first step for solving this question. The idea is that

if we obtain the same (or better) performance using a linear or a quadratic kernel, then there is no need of the cubic polynomial kernel used in the original experiments. If the simplest classifier, linear, is able to perform well then we may observe that there exists a linear relationship between features and that no interactions between them are needed to detect entailment. Furthermore, even if a non-linear relation exists, it is preferable to use a less complex SVM classifier (e.g., quadratic) since it will make easier the analysis of the features that influence the learning model. In this chapter we describe the experiments performed for detecting entailment using new configurations of SVM classifiers and the aspects that were taken into account.

The SVM classifiers used in Baroni, Bernardi, Do and Shan (2012) had the next configuration:

-$(x \cdot y/600)^3$. Polynomial kernel of degree 3, with gamma of 1/600 (1/total number of features) and homogeneous.

-Tolerance of termination criterion ϵ set to 1.6 (this value was tuned on the AN$\models$N data set which was never used for testing)

-Parameter C set to 1 (soft margin parameter/cost)

This type of kernel and parameter values, except ϵ, are the default configuration provided by the used software, i.e., LIBSVM 2.8 (Chang and Lin, 2011) and weka[1]. The effectiveness of SVM depends on the selection of the type of kernel, the kernel's parameters and the soft margin parameter C. The default configuration used in the original experiments seemed to work fine for the aim of the entailment detection task. However, we wanted to try different configurations, especially different degrees of kernels and values of C, not only for trying to improve the accuracy but for facilitating the analysis of the information encoded in the semantic vectors that allows to predict entailment relations. We used LIBSVM 3.13 in C++, the details of the implementations and procedure followed to perform our experiments are mentioned in appendix A.

4.1 Choice of less complex kernel.

As we have explained before, the SVM classifier is fed by a vector containing 600 features. This vector is the concatenation of the semantic vectors representing a Q_1N and Q_2N, respectively. The classifier's output is 1 if Q_1N and Q_2N are in an entailment relation, -1 if they are not. Figure 3.1 shows a graphical representation of the semantic vectors fed to the SVM classifier.

[1]http://weka.wikispaces.com/LibSVM

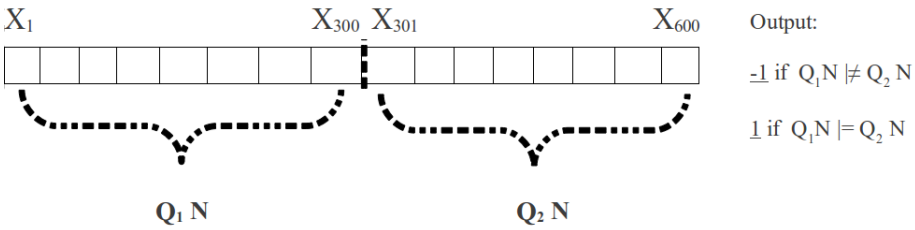


Figure 4.1: Graphical representation of the semantic vectors

Lower degrees of polynomial kernels mean in some sense less degree of complexity. We know that SVM classifiers with polynomial kernels (nonlinear SVMs) are useful to handle linearly inseparable problems by transforming data to a high dimensional space and the higher the degree of the kernels, the more flexible the decision boundary. However, training and testing nonlinear SVMs usually requires more time than a linear SVM classifier. Additionally, high degree polynomial kernels make more difficult the interpretation and the extraction (from the learning model) of the features that were more relevant for deciding if a pair was entailing or not. This is because higher degrees of polynomial kernels imply that the classifier is taking into account more complex combinations between the input features in order to make a prediction.

In section 2.4, it was explained that polynomial SVMs use kernel functions for efficiently compute the dot product in a certain space. The feature space induced by a polynomial kernel depends on its degree. In fact, the feature space induced by a homogeneous polynomial kernel of degree d corresponds to a feature space whose dimensions are spanned by all possible dth-order monomials of the input vector features (Zhang, 2009). For example, an homogeneous kernel of degree 2 induces a feature space spanned by all products of two variables, that is,$\{x_1^2, x_1x_2, x_2^2\}$. The inhomogeneous case is similar but the feature space is spanned by all products of at most two variables, that is, $\{1, x_1, x_2, x_1^2, x_1x_2, x_2^2\}$. For the linear case, which corresponds to a polynomial kernel of degree 1, the feature space corresponds simply to the set of single features, without taking into account products between them for building the classifying function.

For the reasons before described we decided to try three configurations less complex of SVM classifiers:

-$(x \cdot y)$. Linear classifier

-$(x \cdot y/600)^2)$. Polynomial kernel homogeneous with degree 2

-$((x \cdot y/600)+1)^2$). Polynomial kernel inhomogeneous with degree 2

In all cases, the tolerance of termination criterion ϵ remained set to 1.6. But for these classifiers the parameter C was optimized, this process is explained in the next section.

4.2 Optimizing the parameter C.

We have mentioned before that the performance of an SVM classifier is highly dependent on the parameters chosen. The parameter C controls the trade off between allowing training errors and forcing rigid margins. A low value of C creates a "soft margin" that allows some missclassifications, a high value of C increases the cost of misclassifying points and forces the creation of a more accurate model that may not generalize well. The value of C was set to 1 in the original entailment detection experiments. However it is a common practice when using SVMs to find an optimal value of C. In fact, while repeating the experiments with the new classifiers, we soon realized of the need of optimized C values. For example, the linear classifier was not able to converge using the default value C=1 (that worked fine for the original experiments using a cubic kernel). Another important reason to perform the optimization was to be able to properly compare the accuracy among classifiers. Without optimizing the parameters in all the classifiers it is difficult to assert that one type of classifier performs better than other for the same task.

Since our goal was to repeat the experiments SVMpair-out and $\text{SVM}_{\text{quantifier-out}}$ using different classifiers, the C-optimization was performed for each *quantifier-out* and for each *pair-out* (tables 4.1, 4.2). The procedure consisted of performing an "inner" cross validation on each of the training datasets produced by leaving out one quantifier or one quantifer pair (training regimes $\text{SVM}_{\text{quantifier-out}}$ and SVMpair-out explained in section 3.2). Different values of C were tried when doing this cross validation, the optimized C was the one that allowed the best performance. For instance, in the case of the experiment $\text{SVM}_{\text{quantifier-out}}$ where one quantifier is held out as testing data and the remainder is used as training data, cross validations with different values of C were performed on the training data (the dataset that does not contain the quantifier being held out). The same procedure was followed for the experiment SVMpair-out.

The C-optimization process was computationally expensive since the cross-validation had to be performed for each quantifier-out and pair-out and the

overall process had to be performed 3 times (one per each classifier). In order to counteract this, we reduced the number of folds for the cross-validation and the range of values of C to be tried. We used a 5-fold cross-validation and a search range of C $\in$ [0.0001, 0.001 0.005, 0.01, 0.1, 1] for the linear classifier (we stopped at 1 because the classifier was not converging with C's greater than 0.1) and C $\in$ [0.0001, 0.001, 0.01, 0.1, 1, 10,...,100] for the two polynomial classifiers with kernel of degree 2.

The obtained optimized C values are shown in tables 4.1 and 4.2. We can see that the values of the optimized C's using the linear classifier are considerably lower than the optimized C's obtained with the polynomial classifiers. These low C values indicate that it is hard for the linear classifier to find a linear separator that discriminate the data on the respective training datasets so a soft margin where the misclassified examples pay a low cost is more suitable. It is reasonable that the polynomial classifiers allowed a higher value of C, since they map the features to a higher dimension where the training can be discriminated better.

After obtaining the optimized parameters for each of the 3 classifiers the next step was to perform the experiments *SVMpair-out* and $SVM_{quantifier\text{-}out}$ using the new classifier parameter configurations. This stage is described in the next section.

	Linear	Polynomial 1	Polynomial 2
quantifier out	Optimized C	Optimized C	Optimized C
many	0.0001	0.01	0.01
both	0.0001	0.01	0.01
several	0.0001	0.01	0.1
some	0.0001	1	0.01
most	0.001	1	0.1
no	0.0001	0.1	0.1
few	0.0001	0.1	0.1
either	0.0001	0.1	1
much	0.0001	0.1	1
each	0.0001	0.1	0.1
every	0.0001	0.1	1
all	0.0001	0.1	1

Table 4.1: C parameter optimization for $SVM_{quantifier\text{-}out}$. Polynomial 1 is the classifier with homogeneous kernel of degree 2 and Polynomial 2 is inhomogeneous.

	Linear	Poly 1	Poly 2
pair out	Optimized C	Optimized C	Optimized C
many $\models$ several	0.001	0.01	0.1
most $\models$ many	0.001	1	0.1
several $\models$ some	0.0001	1	1
both $\models$ some	0.0001	0.1	0.1
many $\models$ some	0.001	1	1
both $\models$ either	0.0001	1	0.1
every $\models$ many	0.001	1	0.1
much $\models$ some	0.0001	0.1	0.1
all $\models$ most	0.0001	1	1
all $\models$ many	0.0001	1	0.01
each $\models$ some	0.001	1	1
all $\models$ several	0.005	0.1	0.1
all $\models$ some	0.001	1	1
both $\not\models$ several	0.001	1	0.01
both $\not\models$ many	0.0001	0.01	0.01
some$\not\models$many	0.0001	0.1	0.1
many $\not\models$ most	0.0001	0.1	1
many $\not\models$ no	0.001	1	1
few $\not\models$ many	0.0001	0.1	1
either $\not\models$both	0.0001	0.1	0.01
several$\not\models$ few	0.005	1	0.1
few$\not\models$ all	0.0001	0.1	0.1
both $\not\models$most	0.001	0.1	1
some$\not\models$ each	0.0001	1	0.01
Several $\not\models$ all	0.005	0.1	1
several$\not\models$ every	0.0001	0.1	0.1
some$\not\models$ every	0.0001	1	0.01
many $\not\models$every	0.0001	0.1	0.1
many$\not\models$ all	0.001	0.01	1
some$\not\models$ all	0.001	0.1	1

Table 4.2: C parameter optimization for *SVMpair-out*. Polynomial 1 is the classifier with homogeneous kernel of degree 2 and Polynomial 2 is inhomogeneous

4.3 Comparison of performance

The training and testing regimes *SVMpair-out*, $SVM_{quantifier\text{-}out}$ explained in section 3.2 were performed using the exact same datasets but with different classifier's configuration. One linear classifier and two polynomials with quadratic kernels (homogeneous, inhomogeneous) were tried; in this set of experiments the classifiers use a different value of C depending on each *quantifier-out* and *pair-out*. The accuracies obtained with each classifier are shown in table 4.3 and 4.4. Additionally, the confidence interval was calculated in order to be able to compare the different classifiers and identify the cases where the difference in the performance is statistically significant. A 95% confidence interval of a prediction can be computed as (Mitchell, 1997):

$$CI = \pm 1.96\sqrt{\frac{(acc)(1-acc)}{n}}$$

Where *acc* is the accuracy of the prediction and n the number of instances used to compute the accuracy.

4.3.1 Results of $\text{SVM}_{\text{quantifier-out}}$

Table 4.3 shows the obtained results for the experiment $\text{SVM}_{\text{quantifier-out}}$. A quick look shows that the obtained results with the quadratic kernels are very similar to the original ones (table 3.3). In most of the cases they are slightly better. In principle, it seems that there is no need of a cubic polynomial kernel. On the other hand, more dramatic changes in the accuracy can be observed using the linear classifier. If we compare the performance of the linear classifier with the two quadratics (*poly1, poly2*) there are some cases where the performance is apparently a lot better or worst. e.g., the quantifier *few* 78.96% (*linear*) vs 66.38% (*poly1,poly2*). We looked into the cases that had the greatest difference in performance between classifiers, this is, the cases where the *linear* result plus confidence interval is farthest from the *poly, poly2* result minus confidence interval (or vice versa). We looked at the detailed predictions of these cases in order to see if there was some clear pattern that could explain why the linear classifier performs better or worse when detecting entailment involving certain quantifiers.

In $\text{SVM}_{\text{quantifier-out}}$, the cases in which the linear classifier outperformed the quadratic ones with the biggest difference correspond to *few*, *no*, *most*; two of these quantifiers are precisely the only two for which there are not positive examples in the dataset (table 3.3). We have explained before that the training regime $\text{SVM}_{\text{quantifier-out}}$ leaves out all the pairs containing the quantifier in turn

(whether it is in the antecedent or consequent position) and then the resulting model is tested on these pairs containing the quantifier that was left out during the training. We noticed that the performance of the linear classifier was influenced by the amount of training data that remains after taking out the pairs containing the quantifier in turn. For instance, in the case of the quantifier *few*, the classifiers are trained without seeing *few* at all and then they are tested on the instances containing the pairs few $\not\models$ many, few $\not\models$ all, several $\not\models$ few i.e. all the pairs involving the quantifier *few* (table 3.2). By looking at the detailed predictions of $\text{SVM}_{\text{few-out}}$ we realized that the linear classifier predicts correctly a bigger proportion of testing instances containing few$\not\models$many and few$\not\models$all while the quadratic classifiers are better predicting the instances containing several$\not\models$few. This could be happening because even after removing all the pairs containing *few*, the training dataset still has a big amount of positive and negative examples containing the quantifiers *many* and *all* in the consequent position of a pair, while for the quantifier *several* in the precedent position, there are less positive and negative examples remaining in the training dataset; the quadratic classifiers seem to handle better than the linear the prediction of those pairs for which the involved quantifiers were seen little or none in the training data, suggesting that the quadratic classifiers are somehow learning and generalizing the entailment relation. The same pattern can be detected in most of the quantifiers: it seems that the cases where the linear classifier notably outperforms is not really because it is able to generalize better the entailment relation, the performance of the linear classifier is being affected by the unbalanced dataset. We also looked at the cases where the quadratic classifiers outperformed the linear with a greater difference, in these cases we observed again that the amount of data is associated with the failure of the linear classifier, for instance, in the case of the quantifier *either* (table 4.3), when we leave out the pairs involving this quantifier (both$\models$either, either$\not\models$both) the training dataset is left without any positive/negative example of the quantifier *both* appearing in the consequent position, so we observed that the linear classifier predicted considerably less instances containing either$\not\models$both than the quadratic classifiers.

Quantifier	Linear		Poly1		Poly2	
	acc.(%)	CI	acc.(%)	CI	acc.(%)	CI.
each	97.79	±0.8	**98.40**	±0.7	**98.40**	±0.7
every	**97.42**	±0.7	95.62	±0.9	95.51	±1.0
much	85.89	±4.3	**88.31**	±4.0	88.31	±4.0
all	79.80	±1.1	**81.06**	±1.0	80.52	±1.0
several	73.86	±1.5	78.21	±1.4	**79.78**	±1.4
many	77.23	±0.9	**77.49**	±0.9	**77.49**	±0.9
few	**78.96**	±3.7	66.38	±4.3	66.38	±4.3
most	**60.40**	±2.3	58.35	±2.3	58.30	±2.3
some	53.25	±1.2	**55.14**	±1.1	54.11	±1.2
no	**59.10**	±3.6	53.08	±3.7	53.08	±3.7
both	42.16	±2.1	44.26	±2.2	**44.66**	±2.2
either	27.78	±7.8	38.09	±8.5	**41.27**	±8.6
Total	*70.23*	*±0.50*	*71.10*	*±0.50*	*70.93*	*±0.50*

Table 4.3: Results of $\text{SVM}_{\text{quantifier-out}}$. *Poly1* uses an homogeneous quadratic kernel, *Poly2* an inhomogeneous quadratic kernel.

4.3.2 Results of $\text{SVM}_{\text{pair-out}}$

Table 4.4 shows the obtained results for the experiment $\text{SVM}_{\text{pair-out}}$. In the $\text{SVM}_{\text{pair-out}}$ regime, a similar situation to the above described occurs, the obtained results with the quadratic kernels are similar to the original ones (table 3.2) while the linear classifier exhibits more drastic changes. There are several cases in which the linear classifier performed considerably worse than the quadratic classifiers (e.g., *either* $\not\models$ *both, both* $\not\models$ *many, several* $\models$ *some*). The bad behavior of the linear classifier could be influenced by the number of examples in which a quantifier appears in certain position and the label related to it. For example, in the case of *several* $\models$ *some*, if we look to the table 3.2 we can see that most of the times that *several* appears in the antecedent position there is a non-entailment relation, so the linear classifier fails in predicting *several* $\models$ *some* because it generalizes a rule that every time *several* appears in the antecedent position it is likely to be in a non-entailment relation, disregarding which quantifier is in the consequent position.

Quantifier pair	Linear		Poly1		Poly2	
	acc.(%)	CI	acc.(%)	CI	acc.(%)	CI.
all $\models$ some	99.05	±0.6	**99.15**	±0.6	**99.15**	±0.6
all $\models$ several	97.85	±1.2	**98.92**	±0.9	**98.92**	±0.9
each $\models$ some	97.56	±1.2	**98.63**	±0.9	**98.63**	±0.9
all $\models$ many	**92.90**	±1.7	90.49	±1.9	91.41	±1.9
many $\models$ some	80.44	±4.9	**90.01**	±3.7	**90.01**	±3.7
much $\models$ some	87.10	±3.1	**88.71**	±2.9	**88.71**	±2.9
every $\models$ many	**89.35**	±3.1	88.70	±2.9	88.70	±2.9
all $\models$ most	**89.46**	±2.0	83.40	±3.4	84.52	±3.0
several $\models$ some	56.72	±4.0	**75.69**	±3.5	**75.69**	±3.5
both $\models$ some	**70.16**	±3.7	58.46	±4.0	58.46	±4.0
most $\models$ many	**23.76**	±3.4	18.14	±3.1	17.06	±3.0
many $\models$ several	**24.24**	±3.9	16.67	±3.4	21.89	±3.8
both $\models$ either	**6.35**	±6.0	4.76	±5.3	**6.35**	±6.0
Subtotal	*77.44*	*± 0.94*	*77.88*	*± 0.94*	*78.34*	*± 0.93*
some$\not\models$ every	**99.79**	±0.4	**99.79**	±0.4	**99.79**	±0.4
several $\not\models$ all	98.74	±0.9	**99.46**	±0.6	**99.46**	±0.6
some$\not\models$ all	98.96	±1.0	**99.34**	±0.8	98.86	±1.1
several$\not\models$ every	**100.00**	±0.0	99.21	±0.5	99.21	±0.5
few$\not\models$ all	98.09	±1.3	**98.73**	±1.0	**98.73**	±1.0
many $\not\models$every	**98.91**	±0.8	98.26	±1.0	98.26	±1.0
many$\not\models$ all	97.14	±2.6	**97.94**	±2.2	96.33	±2.9
some$\not\models$ each	98.48	±0.8	97.56	±1.0	**98.63**	±0.8
both $\not\models$most	86.45	±3.5	95.39	±2.1	**96.21**	±1.9
several$\not\models$ few	88.11	±5.3	91.61	±4.5	**93.01**	±4.2
both $\not\models$ many	41.04	±4.1	**67.84**	±3.9	**67.84**	±3.9
many $\not\models$ most	54.86	±4.5	**66.31**	±4.3	64.79	±4.4
either $\not\models$both	44.44	±12.3	**60.32**	±12.1	52.38	±12.3
many $\not\models$ no	**58.12**	±3.6	55.04	±3.6	55.04	±3.6
some$\not\models$many	**48.27**	±3.2	47.00	±3.2	47.00	±3.2
few $\not\models$ many	39.13	±7.5	20.50	±6.2	**22.98**	±6.5
both $\not\models$ several	**12.53**	±3.1	11.83	±3.0	8.58	±2.6
Subtotal	*76.84*	*±0.90*	*79*	*±0.87*	*78.66*	*±0.87*
Total	*77.12*	*±0.65*	*78.47*	*±0.64*	*78.51*	*±0.64*

Table 4.4: Results of $\text{SVM}_{\text{pair-out}}$. *Poly1* uses an homogeneous quadratic kernel, *Poly2* an inhomogeneous quadratic kernel.

4.3.3 The optimal classifier

The comparisons between the classifiers performance emphasized the need of a more balanced dataset. The classifiers with a polynomial kernel of degree 2 were in general the most appropriate ones for the task of entailment detection. The linear classifier is apparently able to outperform for some specific quantifiers/pairs but, as we discussed, this is probably just an overfitting effect and the results would change if we modify the distribution of the training and testing dataset. Moreover, if we remember from tables (4.1, 4.2) the optimized C's for the quadratic classifiers were much bigger than the ones obtained for the linear classifier in all cases, this expresses that using the linear features alone the classifier makes a lot of mistakes even for classifying the training instances (that is why the linear classifier needs a small value of C that allows misclassifications). All these aspects suggest that the quadratic classifiers generalize better the entailment relation; the interaction between the features of semantic vectors must be one of the properties involved in the QN entailment, therefore the polynomial classifiers are more suitable for capturing this entailment relation.

Given the evidence that SVM quadratic classifiers are able to capture the entailment relations, the next step would be to identify which are the combinations of features that are playing an important role in this entailment detection. The task of extracting the most significant features and giving them a linguistic interpretation, has been left for future work (see Chapter 5). We conjecture that the analysis of these features will reveal the characteristics of entailment that are encoded in the semantic vectors. In fact, we used a feature selection tool[2] available in LIBSVM, with our quadratic classifiers. The tool assigns a score to each feature, this score measures the impact of each feature in the prediction accuracy of the learning model, a high score indicates high feature relevance. We observed that there was a set of features that were important in practically all the entailment detection cases, while there were features that were important just for specific quantifiers/pairs. This tool combines several selection strategies (e.g., f-score) and it was useful as a preliminary indicator that there is a set of features that is crucial for detecting entailment. However, this tool does not extract the weights of the features from the learning model and the interactions between them, which is something we are interested in. The extraction of the weights from the learning models generated by our experiments and the interpretation of the latent dimensions of our semantic vectors requires further work and analysis.

[2]http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools

4.4 Testing on noun entailment

Finally, a last experiment was performed to confirm that the QN entailment does not generalize to entailment between nouns. We trained a linear and a polynomial (quadratic kernel) classifier on the $Q_1N \models Q_2N$ dataset and test it on the $N_1 \models N_2$ and the $AN \models N$ datasets explained in section 3.1. The results are shown in table 4.5, the accuracy in predicting lexical entailment using the learning model of $Q_1N \models Q_2N$ is not better than the baseline methods (table 3.1). This shows that the information encoded in semantic vectors that allow to detect entailment between quantifiers is not the same one that allows to detect the entailment between nouns. The QN entailment is relying in something different from the feature inclusion that works fine for the lexical entailment. This reinforces the idea of formal semantics that each semantic domain has its own entailment relation.

	Linear		Poly1	
	acc.(%)	CI	acc.(%)	CI
$SVM_{Q_1N \models Q_2N}$ tested on $N_1 \models N_2$	52.38	±0.26	52.74	±0.26
$SVM_{Q_1N \models Q_2N}$ tested on $AN \models N$	48.63	±1.35	52.30	±1.34

Table 4.5: Results of $SVM_{Q_1N \models Q_2N}$. The linear classifier uses a C=0.001 and *Poly1* is an homogeneous quadratic kernel with C=1.

Chapter 5

Conclusions

In this thesis we provided an overview of distributional semantic models as an approach to natural language semantics. In particular, we talked about how distributional representations of words and phrases can be used to detect logical inference patterns like those studied in formal semantics. We described in detail the work of Baroni, Bernardi, Do and Shan (2012) because it is the first one to explore and show that the distributional vectors are able to capture semantic properties of quantifiers and that entailment relations between quantifier phrases can be detected using these representations. Since the entailment detection is done by means of an SVM classifier fed with distributional semantic vectors, we focused on the SVM classifier with the aim of improving the original work. Machine learning literature was reviewed in order to analyze the behavior and the parameters related to this classifying method. Afterwards we realized the need of repeating the entailment detection experiments following a more standard procedure which involved trying first classifiers with less complex polynomial kernels and to perform a parameter optimization. Since these elements were missing in the original experiments we were interested in comparing how much the performance would be affected or benefited by the new configurations of the classifiers.

After performing the new experiments we came to the conclusion that the cubic polynomial classifier used in the original experiments was not really necessary since the classifiers with lower kernel degree (quadratic) were able to perform at least as good as it. The simplest classifier, the linear, showed more dramatic positive and negative changes in the performance, the first impression was that the linear classifier could represent the best option for detecting the entailment of certain quantifiers. However a more detailed analysis (looking at the predictions) revealed that this behavior is probably more related with the distribution of the examples in the dataset than with a real capacity of

the classifier to generalize better. Repeating the experiments with different parameters was useful to analyze some other aspects like the need of a more balanced $Q_1N \models Q_2N$ dataset. The analysis of the soft margin constant C and the detailed predictions among classifiers suggested that the entailment relation between quantifiers involves the interaction between the vector features so a linear classifier which only takes into account the sum of the individual feature influences is not enough.

Finally a training and testing regime was added to the original work, where the learning model that successfully detected entailment between quantifier phrases was tested in semantic vectors representing pairs of nouns in order to detect lexical entailment. As expected, we obtained a bad performance using this regime, reinforcing the idea that the entailment relation that exist between nouns is different from the one between quantifiers; the classifiers are tapping different properties of the vectors.

5.1 Future work

One of the first things that became evident during the analysis of the obtained results was the need of a more balanced $Q_1N \models Q_2N$ dataset, the number of examples per each quantifier pair changes considerably from case to case. A future step would be to extend and balance the dataset and see if the prediction accuracy changes using the new dataset. For instance, in the current results the universal-like quantifiers (each, every, all, much) obtain the best prediction accuracies while the hardest to classify are existential-like ones (some, no, both, either), it would be interesting to see if this pattern remains with a more balanced dataset. The new results could make easier to identify the quantifier pairs for which is harder to detect an entailment relation.

This work represents in some sense a first step to answer the question of what properties encoded in the semantic vectors are decisive for detecting the entailment relations. We showed that a quadratic SVM classifier was able to perform well for the task, this is useful because if we want to extract the information contained in the learning model, it is a lot easier to do it from the model of a quadratic or a linear classifier than extracting it from the model of a higher degree kernel. The obtained learning model can be analyzed to extract those features that have the highest influence for deciding if a pair is in entailment or not, this could reveal interesting linguistic relations. Some considerations must be taken into account like the fact that the current semantic vectors are a reduced version obtained by singular value decomposition, this means that

the meaning of the features cannot be directly interpreted, these latent dimensions represent "topics" so the features would have to be interpreted in terms of these topics.

Appendix A

Data and technology used

The performed experiments in chapter 4 comprised three main stages:
-Corpus preprocessing.
-Building the semantic space.
-SVM classification

We started from the same data used in Baroni, Bernardi, Do and Shan (2012). In the first two stages we used most of the methods and tools used by the cited work. This was done in order to assure that we were recreating the original experimental setup. For the third stage, we wrote all the scripts from scratch since we used a slightly different SVM implementation and different data formats than the ones used in the original experiments. We will mention the involved technology and main considerations in each of these stages.

Corpus preprocessing: The used corpus was a 2.83-billion-token concatenation of the British National Corpus (http://www.natcorp.ox.ac.uk), WackyPedia and ukWaC (http://wacky.sslmit.unibo.it). We were provided with each of these corpora already tokenized, POS tagged and lemmatized. The parsing was performed with the MaltParser (http://www.maltparser.org) and lemmatized with TreeTagger (Schmid, 1995). This is an example of the preprocessed corpus in XML format:

<s>					
How	*How*	*WRB*	*1*	*0*	*ROOT*
does	*do*	*VVZ*	*2*	*0*	*ROOT*
it	*it*	*PP*	*3*	*2*	*SBJ*
affect	*affect*	*VV*	*4*	*2*	*VC*
you	*you*	*PP*	*5*	*4*	*OBJ*
?	*?*	*SENT*	*6*	*2*	*P*
</s>					

The next step was to compute the co-occurrences, i.e., the occurrences of each word in the corpus with other words in the same sentence. It is important to mention that the AN and QN sequences were also considered as single words when computing the co-occurrences. Words with no alphabetical characters, with adjacent or initial dashes, etc., were ignored. After this, the co-occurrences were converted to pointwise mutual information (PMI) scores. The computation of the co-occurrences and the PMI were done using scripts written in Python.

Building the semantic space: Our semantic space was a matrix where the rows contained the most frequent nouns (9.8K), adjectives (8.1K) and also those ANs, QNs and Ns needed for the data sets $Q_1N \models Q_2N$, $N_1 \models N_2$ and $AN \models N$ (explained in chapter 3). The columns contained the content words of the corpus (the most frequent 9.8K nouns, 8.1K adjectives, and 9.6K verbs in the corpus). Each matrix cell was filled with the PMI scores calculated in the previous stage. The result was a sparse matrix with size 48K×27K. The matrix was stored as a text file where each line correspond to a row of the semantic space.

We applied Singular Value Decomposition (SVD) to reduce the dimensionality of the semantic space. This was done by means of a command-line program of the SVDLIBC toolkit[1]. We used the default value of singular vectors to be preserved (300).

SVM classification: Once the semantic space was reduced, the next step was to extract from it our vectors of interest: QNs, ANs and Ns (the vectors needed for the $Q_1N \models Q_2N$, $N_1 \models N_2$ and $AN \models N$ datasets).

For the classification task we used LIBSVM 3.13, a library for support vector machines (software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm). This implementation of support vector machines (Chang and Lin, 2011) has gained wide popularity, it has interfaces and extensions to many programming languages. We didn't use any of the extensions, we decided to use directly the source code of the library in C++. One aspect that must be taken into account is the format of the data. We had to convert our semantic vectors into the LIBSVM format. We show an example of an instance in LIBSVM format, the first element represents the class $\{-1, 1\}$ and the remaining elements represent the value of each feature (feature:value):

[1] http://tedlab.mit.edu/ dr/SVDLIBC/

-1 1:-92.9 2:49.43 3:28.93 4:4.83 ...

LIBSVM allows to set different options when training a classifier, e.g., the svm type, kernel type, degree, soft margin constant, number of folds in cross validation, among others. This flexibility allowed us to incorporate LIBSVM to all our scripts that required an SVM training and testing process.

The scripts that we wrote for performing these experiments were mainly written in Perl (for filtering, splitting datasets) and shell scripting language (for managing the data flow and incorporating LIBSVM). We wrote our shell scripts in such a way that they could work within the Sun Grid Engine[2] (SGE). This engine is used in clusters and it is responsible for accepting, scheduling, dispatching, and managing the remote and distributed execution of large numbers of standalone, parallel or interactive user jobs. We were allowed to use a cluster of the University of Trento. Running our scripts in a cluster was convenient since our experiments (parameter optimization, $\text{SVM}_{pair-out}$, $\text{SVM}_{quantifier-out}$) involved splitting the dataset into many subsets. The training and testing procedures had to be performed for each of these subsets with different configurations according to the experiment. Without the use of a cluster, the running time would have been prohibitive.

Attachments: As an attachment to this thesis, we provide the lists of words that we extracted from the corpus to build the semantic space:

- matrix_rows.txt: Contains the concatenation of the most frequent 9.8K nouns, 8.1k adjectives, the AN and QN phrases used in the experiments and the adjectives, quantifiers and nouns contained in those sequences.
- matrix_columns.txt: Contains the concatenation of the most frequent 9.8K nouns, 8.1K adjectives, and 9.6K verbs in the corpus.
- noun_hypernyms.txt: Contains the 9734 hyponym-hypernym pairs used for building the $N_1 \models N_2$ dataset.
- q-pairs.txt: Contains the fixed list of entailing and non-entailing quantifier pairs
- svdout_QN_sample.mat: Contains a sample of the first 1K rows of our reduced semantic space, i.e., the semantic vectors.

Additionally, we attach the scripts that perform the training and testing phases in order to show the usage and parameters of LIBSVM.

[2]http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html

List of Tables

Bibliography

[1] Alpaydin E. (2004). Introduction to Machine Learning. The MIT Press, October 2004, ISBN 0-262-01211-1

[2] Baroni M., Bernardi R., Do N. and Shan Ch. Entailment above the word level in distributional semantics. In Proceedings of EACL. 2012.

[3] M. Baroni and A. Lenci. 2011. How we BLESSed distributional semantic evaluation. Proceedings of the EMNLP 2011 Geometrical Models for Natural Language Semantics (GEMS 2011) Workshop, East Stroudsburg PA: ACL, 1-10.

[4] Baroni, M. & Lenci, A. (2010). Distributional Memory: A general framework for corpus-based semantics. Computational Linguistics 36 (4): 673-721

[5] Baroni M. and Zamparelli R. 2010. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In Proceedings of EMNLP, pages 1183-1193, Boston, MA.

[6] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, 5th Annual ACM Workshop on COLT,pages 144-152, Pittsburgh, PA, 1992. ACM Press.

[7] Barwise, Jon and Robin Cooper. 1981. Generalized quantifiers and natural language. Linguistics and Philosophy

[8] Bhagat R., Pantel P., and Hovy E. 2007. LEDIR: An unsupervised algorithm for learning directionality of inference rules. In Proceedings of EMNLP-CoNLL.

[9] Burges, C. J. C. 1998. A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery 2: 121167.

[10] Cherkassky, V., and F. Mulier. 1998. Learning from Data: Concepts, Theory,and Methods. New York: Wiley.

[11] Chierchia, Gennaro, and Sally mcconnell-Ginet (2000) Meaning and Grammar: An Introduction to Semantics, 2nd ed. MIT Press

[12] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2(3):27:127:27.

[13] Church, K. W. and Hanks, P. (1989) Word association norms, mutual information and lexicography. In Proc. 27th Annual Meeting of ACL, Vancouver. 1989: 76-83

[14] Cortes, Corinna; and Vapnik, Vladimir N.; "Support-Vector Networks", Machine Learning, 20, 1995

[15] Dagan, I., and Glickman, O. 2004. Probabilistic textual entailment: generic applied modeling of language variability, In PASCAL Workshop on Learning Methods for Text Understanding and Mining, Grenoble, France.

[16] Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., & Harshman, R. A. (1990). Indexing by latent semantic analysis. Journal of the American Society for Information Science (JASIS), 41 (6), 391-407.

[17] Erk, K., & Pado, S. (2008). A structured vector space model for word meaning in context. In Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP-08), pp. 897-906, Honolulu, HI

[18] Evert, Stefan (2010). Google Web 1T5 n-grams made easy (but not for the computer). In Proceedings of the 6th Web as Corpus Workshop (WAC-6), Los Angeles, CA

[19] Firth, J. R. (1957). A synopsis of linguistic theory 1930-1955. In Studies in Linguistic Analysis, pp. 1-32. Blackwell, Oxford.

[20] Frege Gottlob .(1980). "ber Sinn und Bedeutung" in Zeitschrift fr Philosophie und philosophische Kritik 100: 25-50. Translation: "On Sense and Reference" in Geach and Black

[21] Geffet M. and Dagan I. 2005. The distributional in- clusion hypotheses and lexical entailment. In Pro- ceedings of ACL.

[22] Golub, G. H., & Van Loan, C. F. (1996). Matrix Computations (Third edition). Johns Hopkins University Press, Baltimore, MD.

[23] Harris, Z. (1954). Distributional structure. Word, 10 (23), 146-162

John Lyons. Linguistic Semantics: An Introduction. Cambridge, U.K.: Cambridge University Press. 1995.

[24] Kotlerman , Dagan Ido , Szpektor Idan, and Zhitomirsky-Geffet Maayan. 2009. Directional distributional similarity for lexical expansion. In Proceedings of the ACL-IJCNLP 2009 Conference Short Papers (ACLShort '09). Association for Computational Linguistics, Stroudsburg, PA, USA, 69-72.

[25] Lin, D. (1998). Automatic retrieval and clustering of similar words. In roceedings of the 17th international conference on Computational linguistics, pp. 768-774. Association for Computational Linguistics.

[26] Liu, H., and Setiono, R., Chi2: Feature selection and discretization of numeric attributes, Proc. IEEE 7th International Conference on Tools with Artificial Intelligence, 1995, 338-391.

[27] Lund, K., & Burgess, C. (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. Behavior Research Methods, Instruments, and Computers, 28 (2), 203-208

[28] Manning C., Raghavan P., Schütze H., An Introduction to Information Retrieval, Cambridge University Press. 2008.

[29] Mitchell Jeff and Lapata Mirella. 2010. Composition in distributional models of semantics. Cognitive Science, 34(8):13881429.

[30] Mitchell. Tom M. (2006). "The Discipline of Machine Learning." Machine Learning Department technical report CMU-ML-06-108, Carnegie Mellon University.

[31] Mitchell. Tom M. Machine Learning,McGraw Hill, 1997.

[32] Pado, S., & Lapata, M. (2007). Dependency-based construction of semantic space models. Computational Linguistics, 33 (2), 161-199

[33] Schmid Helmut. 1995. Improvements in part-of-speech tagging with an application to German. In Proceedings of the EACL-SIGDAT Workshop, Dublin, Ireland.

[34] Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation, 28 (1), 11-21.

[35] Szpektor I. and Dagan I. 2008. Learning entailment rules for unary templates. In Proceedings of COLING.

[36] Turney, Peter. 2006. Similarity of semantic relations. Computational Linguistics, 32(3):379416

[37] Turney and P. Pantel. 2010. From frequency to mean- ing: Vector space models of semantics. Journal of Ar- tificial Intelligence Research, 37:141188.

[38] Vapnik, V. 1995. The Nature of Statistical Learning Theory. New York: Springer.

[39] Voorhees E. M. and Harman D. K. , editors. 1999. The Seventh Text REtrieval Conference (TREC-7), volume 7. NIST.

[40] Weaver, W. (1955). Translation. In Locke, W., & Booth, D. (Eds.), Machine Translation of Languages: Fourteen Essays. MIT Press, Cambridge, M

[41] Weeds J., Weir D., and McCarthy D. 2004. Character- ising measures of lexical distributional similarity. In Proceedings of COLING

[42] Wittgenstein, L. (1953). Philosophical Investigations. Blackwell. Translated by G.E.M. Anscombe

[43] Zhang, Ming . "Artificial Higher Order Neural Networks for Economics and Business." IGI Global, 2009. 1-542. Web. 27 Aug. 2012. doi:10.4018/978-1-59904-897-0