7-25-2011

Measuring Semantic Similarity: Representations and Methods

Mihai Cosmin Lintean

To the University Council:

The Dissertation Committee for Mihai Cosmin Lintean certifies that this is the final approved version of the following electronic dissertation: "Measuring Semantic Similarity: Representations and Methods."

Vasile Rus, Ph.D.
Major Professor

We have read this dissertation and recommend its acceptance:

Arthur Graesser, Ph.D.

King-Ip (David) Lin, Ph.D.

Vinhthuy Phan, Ph.D.

Accepted for the Graduate Council:

Karen D. Weddle-West, Ph.D.
Vice Provost for Graduate Programs

MEASURING SEMANTIC SIMILARITY: REPRESENTATIONS AND METHODS

by

Mihai Cosmin Lintean

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Major: Computer Science

The University of Memphis

August 2011

DEDICATION

To family and friends ...

... especially to my dear parents and my little brother.

ACKNOWLEDGEMENTS

My deepest appreciation goes to my advisor, Dr. Vasile Rus, for his patience with me and for giving me the right input and support to keep me going. People say writing a dissertation is not an easy thing. For me, it was one of the most difficult things I had to do in my life so far. I do not think there are words which can express the amount of gratitude that I have for my advisor, without whom none of this would have been possible. Thank you, Vasile. You have been a wonderful advisor and a great mentor. I will never forget all that you and your family did for me.

I would also like to thank a few other people who had a direct positive influence on me and my dissertation: Dr. Art Graesser, whom I admire very much, Dr. Mark Conley, for his useful advice on writing, Dr. David Lin, for being such a wonderful tutor and professor, and Zhiqiang Cai, for being a great colleague and friend.

ABSTRACT

Lintean, Mihai Cosmin. Ph.D. The University of Memphis. August, 2011. Measuring Semantic Similarity: Representations and Methods. Major Professor: Vasile Rus, Ph.D.

This dissertation investigates and proposes ways to quantify and measure semantic similarity between texts. The general approach is to rely on linguistic information at various levels, including lexical, lexico-semantic, and syntactic. The approach starts by mapping texts onto structured representations that include lexical, lexico-semantic, and syntactic information. The representation is then used as input to methods designed to measure the semantic similarity between texts based on the available linguistic information. While world knowledge is needed to properly assess semantic similarity of texts, in our approach world knowledge is not used, which is a weakness of it. We limit ourselves to answering the question of how successfully one can measure the semantic similarity of texts using just linguistic information. The lexical information in the original texts is retained by using the words in the corresponding representations of the texts. Syntactic information is encoded using dependency relations trees, which represent explicitly the syntactic relations between words. Word-level semantic information is relatively encoded through the use of semantic similarity measures like WordNet Similarity or explicitly encoded using vectorial representations such as Latent Semantic Analysis (LSA). Several methods are being studied to compare the representations, ranging from simple lexical overlap, to more complex methods such as comparing semantic representations in vector spaces as well as syntactic structures. Furthermore, a few powerful kernel models are proposed to use in combination with Support Vector Machine (SVM) classifiers for the case in which the semantic similarity problem is modeled as a classification task.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

HTML	HyperText Markup Language
IR	Information Retrieval
IDF	Inverse Document Frequency
ITS	Intelligent Tutoring System
LSA	Latent Semantic Analysis
LDA	Latent Dirichlet Allocation
MSR	Microsoft Research (Paraphrase) Corpus
NLP	Natural Language Processing
PKA	Prior Knowledge Activation
POS	Part-of-speech
RTE	Recognizing Textual Entailnment
QA	Question Answering
SVM	Support Vector Machine
T2T	Text-to-text
TF	Term Frequency
W2W	Word-to-word
WN	WordNet
ULPC	User Language Paraphrase Corpus
XML	Extensible Markup Language

CHAPTER 1

INTRODUCTION

1.1. Introduction

This dissertation addresses the challenging task of measuring semantic similarity between texts written in natural language. The task consists of qualitatively and quantitatively assessing how close in meaning two texts are. The length of the texts on which similarity is assessed, can vary from single words to phrases and sentences, to paragraphs and even entire documents.

Measuring semantic similarity is a central problem in Natural Language Processing (NLP) due to its importance to a variety of applications ranging from web-page retrieval (Park, Ra, and Jang 2005), question answering (Ibrahim, Katz, and Lin 2003; Rinaldi et al. 2003), word sense disambiguation (Patwardhan, Banerjee, and Pedersen 2003), text classification (Lodhi et al. 2002) and clustering, to natural language generation (Iordanskaja, Kittredge, and Polguere 1991) and conversational agent/dialogue system in intelligent virtual tutoring (Graesser et al. 2005; McNamara, Boonthum, and Millis 2007). For example, in Intelligent Tutoring Systems (ITSs), students are sometimes required to respond in free form natural language to specific tasks given by the virtual tutor. These free form answers must then be automatically analyzed for accuracy. The typical approach is to compare how similar they are in relation to a list of ideal answers written in natural language, usually handcrafted by human experts. As another example, in Information Retrieval (IR) systems, the search queries created by users are in the form of small sets of keywords or phrases that may include content words such as verbs or nouns along with some function words such as prepositions. The IR system is supposed to return documents that are semantically similar to the query.

Assessing the semantic similarity of texts is a rather general problem, which can take many forms. Some of the most commonly studied in the literature are the tasks on qualitatively assessing the existence of semantic similarity relations between texts. In particular, research focused on two major types of semantic similarity relations: *paraphrase* and *entailment*. In the following, we exemplify and discuss on these relations, and then later on, in Section 1.7, we will present some practical applications where, being able to detect such relations can be very useful.

The goal of paraphrase identification is to determine whether two given texts convey the same meaning. Consider the following pair of two sentences from the Microsoft Research (MSR) Paraphrase Corpus (Dolan, Quirk, and Brockett 2004), in which Text A is labeled as a paraphrase of Text B and vice versa:

> Text A: *Ricky Clemons' brief, troubled Missouri basketball career is over.*
>
> Text B: *Missouri kicked Ricky Clemons off its team, ending his troubled career there.*

Both texts convey the fact that Ricky Clemons had a troubled career at Missouri and that now it is over (or it ended). At a closer look, we notice that each sentence also contains extra information, which is not present in the other. Text A mentions that Ricky Clemons' career was brief and that he played basketball during his career. Text B mentions how Ricky's career ended: he was kicked off its team. The annotators of this example decided that, in most part, the two sentences convey the same message and the differences are minor enough to be ignored. It is obvious that this decision is subjective, and the reader can agree or disagree with the annotators' decision to label the two sentences as paraphrases.

Although paraphrase identification has been defined qualitatively in the form of a binary classification task consisting of detecting the presence or absence of semantic similarity, which is somehow understandable given the vagueness of the definition of a paraphrase, we redefine it as a quantitative task of measuring semantic similarity between the corresponding texts. We quantify the paraphrase relations on a normalized scale from 0 to 1, where 1 means that the two input texts are semantically equivalent, while 0 means that they are not. Alternatively, the paraphrase relation can be quantified on a scale from -1 to 1, where 1 means that the two sentences share the same meaning and nothing else (i.e., "*John heard Mary running*" versus "*Mary was heard by John while she was running*"), while -1 means the sentences have opposite or contradictory meanings (i.e., "*I ate John's apple*" versus "*I didn't eat John's apple*"). Sometimes, the semantic similarity scale does not require a normalization factor. Supposedly, we can define a semantic similarity metric around a borderline value, which is usually 0, where all the values below the borderline state that the input samples are semantically different, while all values above the borderline state that the input samples are semantically close. The further away from the borderline these values are, the more confident the statements are. This idea of a borderline threshold can be applied to the 0-1 normalization range allowing a binary classification approach to paraphrase identification. The threshold can be used to classify all instances as paraphrases, when the quantitative similarity score exceeds the threshold value, or non-paraphrases when the similarity score is below the threshold.

Another example of qualitatively assessing the semantic similarity between texts is the task of recognizing textual entailment. In this task, the challenge is to determine whether one text is (or is not) entailed from another. One example of such an entailment relation is given by the following pair of sentences extracted from the Recognizing Textual Entailment (Dagan, Glickman, and Magnini 2005, RTE) corpus, where T is considered the entailing ”Text” and H the entailed ”Hypothesis”:

> T: *There are also tanneries, sawmills, textile mills, food-processing plants, breweries, and a film industry in the city.*
>
> H: *Movies are also made in the city.*

We say that *T entails H* if the meaning of *H* can be inferred from the meaning of *T* (Dagan, Glickman, and Magnini 2005). In this example, the entailed Hypothesis can be easily construed from the entailed Text. The Hypothesis contains no extra information relative to the Text. This is very different from the paraphrase example given earlier, where both texts contained additional information. In the case of entailment, the fact that the Text contains extra information does not affect the relation of entailment. The particular case of entailment in which there is no extra information in the Text with respect to the Hypothesis, the Text-Hypothesis pair are in a paraphrase relation. That is, paraphrase can be viewed as bidirectional entailment between two input instances, T and H, where *T entails H* and *H entails T* (Rus et al. 2008a).

Textual entailment is strongly related to the relation of logical entailment. In current literature we often see a tendency to intermingle these two notions of textual and logical entailment. In their survey of paraphrasing and textual entailment methods, Androutsopoulos and Malakasiotis (2010) describe textual entailment in terms of logical entailment, and paraphrasing as a particular case of bidirectional entailment relation. We argue that textual paraphrasing and entailment should have looser definitions since they rely on natural language which is often ambiguous. Minor inconsistencies, differences or extraneous information should be permitted as long as the main message stays the same. Therefore, in this dissertation we regard textual entailment as a semantic relation between two texts which does not necessarily follow strict logical thinking to support or reject the assumed inferences in the texts. For example, in current stage of this research, it is not expected from a system that does textual entailment recognition to support arithmetic reasoning, which would help to correctly classify the following pair or sentences as a positive instance of textual entailment:

T: *Jimmy has 2 apples and 3 oranges in his basket.*

H: *Jimmy has 5 pieces of fruit in his basket.*

Given the loose definitions of paraphrase and entailment relations, an important question to ask is with respect to how much extra information in one text (or Hypothesis for the case of entailment) can be ignored in order to detect the presence of a paraphrase (or entailment) relation. Defining a quantitative semantic similarity metric first could help find an answer by measuring the level of extra information that could be ignored.

There are other qualitative semantic relations that pertain to the general task of semantic similarity assessment. One such example is to detect the *elaboration* of a text (McCarthy et al. 2008), where text *B* is said to be an elaboration of text *A* if *B* elaborates on the topic presented by *A*, i.e., adds more details about the topic that is discussed. In some cases, elaboration can be viewed as reverse entailment, and vice versa. We will explore in this proposal automatic methods to assess the semantic similarity relations of paraphrasing and entailment.

The task of assessment or *recognition* of semantic similarity relations, such as paraphrase or entailment, is closely related to two other tasks: the *extraction* and *generation* of pairs (or groups) of texts in which such semantic relations are reflected. As noted in (Androutsopoulos and Malakasiotis 2010), the distinction between these three tasks is not always clear in the literature. Paraphrase or entailment *extraction* (Barzilay and Lee 2003; Brockett and Dolan 2005; Madnani and Dorr 2010) is the task of extracting from various sources fragments of texts that are in a paraphrase or entailment relation. For instance, paraphrase relations could be extracted from texts containing redundant semantic content such as news articles from different media sources, which cover the same topic, or from multiple English translations, made by different translators, of the same text, e.g., the Bible. In the case of paraphrase or entailment *generation* one input text is given and the task is to generate one or more texts which are semantically related to the given text. As a concrete example of paraphrase generation, we mention the work of McKeown (1983) which describes the paraphrase component for a natural language question-answering system. In this system, the user's questions are paraphrased and presented to the user for validation, before the questions are evaluated and answered.

1.2. Goal

The overarching goal of this dissertation is to explore, investigate, and propose ways to quantify and measure semantic similarity between texts. The general approach is to rely on linguistic information at various levels, including lexical, lexico-semantic, and syntactic. The approach starts by mapping texts onto structured representations that include lexical, lexico-semantic, and syntactic information. The representation is then used as input to methods designed to measure the semantic similarity between texts based on the available linguistic information. While world knowledge is needed to properly assess semantic similarity of texts, in our approach world knowledge is not used, which is a weakness of it. We limit ourselves to answering the question of how successfully one can measure the semantic similarity of texts using just linguistic information. The lexical information in the original texts is retained by using the words in the corresponding representations of the texts. Syntactic information is encoded using dependency relations trees, which represent explicitly the syntactic relations between words. Word-level semantic information is relatively encoded through the use of semantic similarity measures like WordNet Similarity or explicitly encoded using vectorial representations such as Latent Semantic Analysis (LSA). We plan to study several methods to compare the representations, ranging from simple lexical overlap, to more complex methods such as comparing semantic representations in vector spaces as well as syntactic structures. Furthermore, we propose a few kernel models which can be used in combination with Support Vector Machine (SVM) classifiers for the case in which the semantic similarity problem is modeled as a classification task. Although SVM classifiers are mostly used in binary classifications, there are ways to extend and normalize their output such that it can be used as a quantitative measure of similarity instead of a simple binary decision.

1.3. Semantic Similarity: From Words to Texts

The concept of semantic similarity is a rather fuzzy concept in the literature. To better understand semantic similarity, it is important to analyze the related concept of semantic relatedness. Semantic similarity and relatedness were used interchangeably at word level(Pedersen, Patwardhan, and Michelizzi 2004;

Mihalcea, Corley, and Strapparava 2006), which may confuse some readers. While semantic similarity is a close-connection between the meaning of two words or texts, semantic relatedness refers to remote semantic relations. In the following, we explain the difference between these two concepts, starting at word level and then moving on to larger texts, e.g., sentences or paragraphs.

Finding the semantic similarity between two words can be defined in terms of how interchangeable the two words are in context. In other words, the following question can be asked: what is the extent to which two words can be used interchangeably in a text without changing its meaning? Synonymy and hyponymy are such examples of semantic similarity relations at word level (Jurafsky and Martin 2002). Two words are in a synonymy relation if they can be interchanged in some (but not necessarily all) contexts (e.g., *confused* and *bewildered*). For hyponymy, one word is the hyponym of another if its meaning is an instance of one of the broader meaning of the other word (e.g., *dog* is hyponym of *animal*). Synonymy is a symmetric relation (bidirectional), while hyponymy is asymmetric (unidirectional). The inverse relation for hyponymy is called hypernymy (i.e., animal is hypernym to dog).

Semantic relatedness refers to words being used in the same context without being interchangeable. Semantic relatedness can be regarded as a remote semantic relation between words. For instance, *dog* and *mutt* are similar while *dog* and *bark* are related. Other known word-level semantic relations are antonymy, describing opposite meanings (i.e., *hot* vs. *cold*), and meronymy, when one word meaning describes a part of the concept defined by the other word (i.e., *nucleus* is part of *eukaryotic-cell*).

There is another, more operational, difference between semantic similarity and semantic relatedness, at word level. For semantic similarity the words we

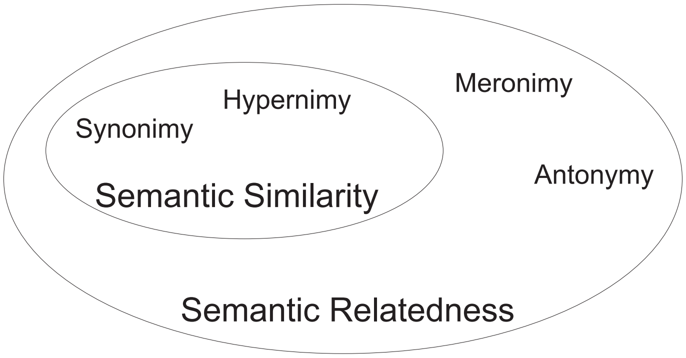


Figure 1.1
Word level: Semantic Similarity vs. Semantic Relatedness

compare are required to have the same part of speech, while for semantic relatedness the words can actually have different parts-of-speech (i.e., *compete*-verb vs. *competition*-noun, *food*-noun vs. *edible*-adjective). Figure 1.1 depicts our view on the distinction between the two measures of semantic similarity and relatedness at the word level. Therefore, we view the notion of semantic similarity at word level as nothing more than a particular case of semantic relatedness.

We now convert this view into a normalized metric of semantic similarity between words. We define a metric $Word_{Sim}$ with values ranging from 0 to 1, measuring the semantic relationship between two given words, W_A and W_B, in the following way:

- a value equal or close to 0 defines the degree to which W_A and W_B are unrelated. The lower this value is, the higher the confidence that the words are unrelated. A $Word_{Sim}(W_A, W_B) = 0$ value means that the two words are completely unrelated, and there is no correlation between the usage of the two in the written or spoken language of the words.

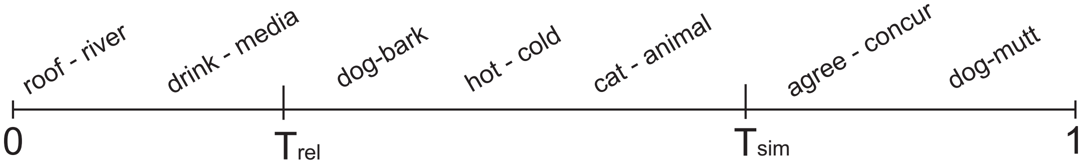


Figure 1.2
A Normalized Word-to-Word Semantic Similarity Metric

- a value equal or slightly greater than a threshold value, T_{rel}, defines the degree to which W_A and W_B are semantically related, but not similar in meaning. The higher this value is, the higher the degree of relatedness between the words.
- a value equal or greater than a threshold value, T_{sim}, defines the degree to which W_A and W_B are semantically similar, or share, more or less, the same meaning. The higher this value is, the higher the degree of semantic similarity between the words. A $Word_{Sim}(W_A, W_B) = 1$ value means that the meaning of the words is exactly the same, and therefore these words can be used interchangeably in any context whatsoever.

Figure 1.2 gives a graphical representation of our previously defined metric, with some representative examples. Our definition of the semantic similarity at word-based level is inspired by recent work on word-to-word semantic similarity metrics (Hirst and St-Onge 1998; Banerjee and Pedersen 2003; Patwardhan 2003; Pedersen, Patwardhan, and Michelizzi 2004; Landauer et al. 2007). One issue with these word-based similarity metrics is that finding the boundaries necessary to decide word relatedness or similarity can be very challenging as the boundaries can be different for different tasks and metrics. Chapter 4 details the word-to-word similarity metrics.

The just described rubric for scoring relatedness does not apply to words that are opposite in meaning (e.g., *hot* vs *cold*). Obviously they are not semantically similar, but there is however a strong semantic relation between them. A new rubric should be defined in this case on a normalized scale between -1 and 1, where values equal or close to -1 reflect that the two words are in a relation of antonymy (i.e., *agree* vs. *disagree*), while values equal or close to 1 reflect a relation of synonymy (i.e., *agree* vs. *concur*). The research work presented in this dissertation will focus only on the rubric.

The above definition of semantic similarity and relatedness can be extended to longer texts. In general, we say that two texts are similar if they convey more or less the same meaning. Examples of textual relations that can be regarded as semantic similarity relations include the relations of paraphrase. The two texts would be related if they are remotely related, e.g., discussing different aspects of same event or topic. Examples of semantically related texts are the relation between a question and its answer or the relation of causality between two texts describing the cause and effect of some phenomenon or events, where one event occurs as a consequence of the other.

Similarly, as for the word-based level, we can define a normalized semantic similarity metric between two texts. The length of the texts can range from syntactic phrases or full sentences to paragraphs or documents. Figure 1.3 depicts this metric, showing some representative examples of relations between two sentences or paragraphs.

This dissertation explores the concepts of semantic similarity and relatedness between texts the size of a sentence only. Some of the methods that we study make use of the semantic similarity and relatedness at word level.

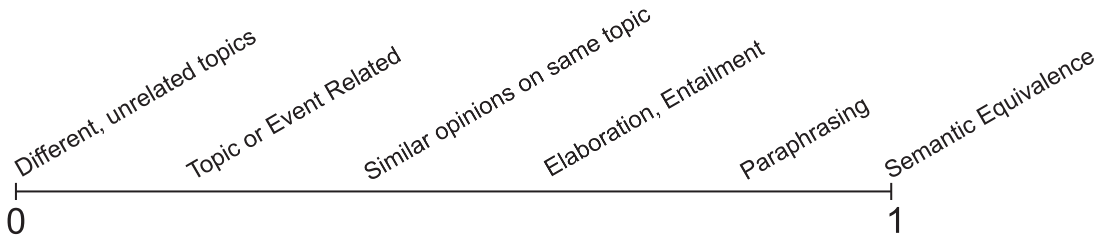


Figure 1.3
A Normalized Text-to-Text Semantic Similarity Metric

1.4. Challenges of Measuring Semantic Similarity at Sentence Level

When one analyzes the problem of semantic similarity between two sentences at a more conscientious level, it becomes obvious that the problem is extremely challenging, even for humans. More times than expected, the context of the given text instances being assessed is not well defined and the topic described by the input sentences is too vague to correctly establish whether there is a semantic similarity relation between them or not. Consider the following two sentences: *"John bought an apple."* and *"John purchased an apple"*. Since *buy* and *purchase* represent the same action, we can say that the two sentences are semantically similar. But what if there is the case where the apple in the first sentence is a fruit, while the apple in the second sentence is actually a computer? So, because not enough information about the sense of the word "apple" has been given, we can easily give a faulty answer to an apparently very simple problem. The sense of the words could have been easily induced, if we had f being used, e.g., *"While passing by a grocery store, John bought an apple"* and *"He needed a new computer, one that was faster and more reliable than his old laptop, so John purchased an apple."* This example suggests that, before going into the task of comparing the two sentences, we first need to determine the exact sense of each word that is used in the text.

This step is known in the NLP research community as the task of *word sense disambiguation*, which is also a challenging and highly researched task (Patwardhan, Banerjee, and Pedersen 2003).

But even when we know the exact senses of the words, the task of, for instance, paraphrase identification does not become any easier. Consider the following example:

Text A: *That information was first reported in today's edition of NY Times.*

Text B: *The information was first printed yesterday in the NY Times.*

In this example, we assume the sentences refer to the same news and the same newspaper. To correctly identify the presence of a paraphrase relation, we need to make some inferences and to have knowledge about the newspaper business. The critical information we need to know is that the NY Times newspaper usually prints the paper one day before distributing it. Once we know this additional information, we can correctly infer that the two sentences are indeed paraphrases. Having general world knowledge as well as domain specific knowledge, e.g., newspaper business, and being able to make logical deductions or inferences based on that knowledge is a very challenging task, in fact one of the most challenging goals of the broader Artificial Intelligence research area.

We finish our debate on the difficulty of measuring semantic similarity by restating one of the previous examples given in the introductory section, about the need to have arithmetic reasoning, on some particular cases of paraphrasing or entailment. Consider the following two sentences: *"John bought 3 apples and 2 pears."* and *"John bought 5 pieces of fruit"*. Being able to reason with numbers and arithmetic operations, one could potentially consider these sentences as being paraphrases, in a more loosened context, or being in a unidirectional relation of

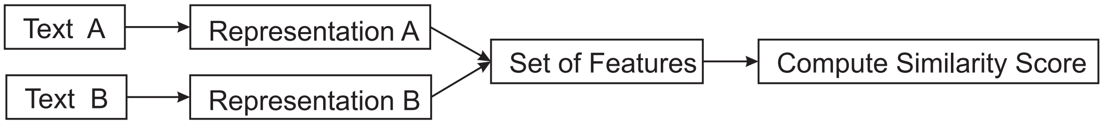


Figure 1.4
Computing Textual Semantic Similarity - Major Steps

entailment, from the first sentence to the second. In our investigation, we assume that we do not have access to the large context of the given input sentence, nor that we have a computational resources encoding general world knowledge. As already mentioned, we limit ourselves at exploring the role of linguistic information on measuring similarity and relatedness between texts the size of a sentence and using these measurements to establish the existence of semantic similarity relations such as paraphrase or entailment.

1.5. The Proposed Framework for Semantic Similarity Assessment

To measure the semantic similarity, we propose a three phase process. In the first phase, we convert the input texts into semantic representations which contain all the necessary information needed for the next steps. In the second phase, we extract sets of features by comparing various lexico-semantic items, which are contained in the semantic representations of the two texts. The extracted features are then used to compute a semantic similarity score between the initial given texts. Figure 1.4 graphically summarizes the above mentioned.

1.5.1 A Semantic Representation of Texts

For the semantic representations (SR) of the texts we propose the following formalism:

SR: ($Word, Lemma, POS, Specificity,$ WN-SENSE|LSA-Vector,

$$(< - : dep_{type} : dep_{mod} > | < dep_{head} : dep_{type} : - >)+)+$$

The above structured representation includes items that correspond to lexical, semantic, and syntactic elements in the original text. That is, the original text is regarded as an ordered list of lexical tokens, which are mostly represented by words. A token has several attributes (most of these make sense when the token represents a word in the text): the initial lexical form of the token, the lemma form of the word, the word's part-of-speech (POS), the weighted specificity of the word (words that are specifics to certain topics and rather uncommon in general, are given more importance by this value), a semantic representation for the sense of the word - either through WordNet's semantic senses or the LSA's vectorial representations - and finally, a list of syntactic dependencies with the other words in the same sentence. Here is a concrete example[1] of how to map a given short sentence into the representation which was formalized above. Given the following sentence: *"Peter went to Seattle last Thursday."*, we represent this sentence as follows:

[(Word=$Peter$, lemma=$peter$, POS=NNP, WN_{SENSE}=1,Deps=($went$:nsubj:-)),
($went$, go, VBP, 1, (-:nsubj:$peter$; -:prep_to:$seattle$; -:tmod:$thursday$)),
(to, to, N, 1, ()),
($Seattle$, $seattle$, NNP, 1, ($went$:prep_to:-)),
($last$, $last$, JJ, 1, ($thursday$:amod:-)),
($Thursday$, $thursday$, NNP, 1, ($went$:tmod:-; -:amod:$last$))]
(., ., PERIOD, 1, ())]

[1] We omit to show the weighted specificity of words in this example, since these are float values that depend a lot on the type of weighting that is used.

More details on this representation are further given in section 2.4. The main idea behind this representation is that we want to have easy access to any pertainable lexical, syntactic and semantic information about the text. The representation needs to allow for easy use of simple string matching operations that can be performed in the next steps. All of the information stored in such a representation can be automatically derived using state-of-the-art methods which in late years have become mature enough for practical use. The performance score of these tools are fairly good, assuring us that the data they produce is accurate enough for our task. Section 2.5 gives more details on this data extraction phase.

1.5.2 Comparing the Semantic Representations

Once the input texts have been converted into their appropriate semantic representations, we can proceed with comparing these representations of texts and extract meaningful features which will help in computing the similarity metric. Comparing two texts by their semantic representations can be done in many ways. In the simplest case, one can measure the overlapping score of lexical tokens between the two texts. For example, suppose we want to compare the sentence given in previous subsection (A) with another sentence (B), which basically conveys the same message but with more details:

A. *Peter went to Seattle last Thursday.*

B. *Last Thursday, my friend Peter flew to Seattle for a business meeting.*

Sentence A has 7 lexical tokens (including the ending period), while sentence B has 14 lexical tokens (including the comma after *Thursday*). If we compare the original lexical form of all tokens and we ignore case (i.e., upper case letters) we find that there are 6 common tokens (including the ending period). We normalized this number by dividing it to the average number of tokens in a sentence (i.e., $(7 + 14)/2 = 10.5$) and we get a simple lexical-based similarity score of 0.57, between the two input sentences.

To compare two input texts through lexical overlap metrics, we can do this at unigram level, as shown before, or at bigram level, to account for some syntactic relations that can appear between consecutive words. In addition, tokens can be compared by using their word form or the lemma form, the POS or their associated semantic-based vectorial representation (e.g., LSA vectors). Also, to compare two tokens, we can employ various semantic similarity measures at word level. To account for the syntactic aspect, we can compare the lists of the extracted syntactic dependencies, or their corresponding syntactic trees. In the following chapters we present experiments on many assortments of these comparing approaches.

1.5.3 Computing Semantic Similarity

Based on the set of features extracted, various semantic similarity measures can be computed. Function based models, such as logistic regression, or support vector machines, can be trained from expert-labeled datasets (gold standards) that are specific to a particular task of semantic similarity. In most cases that we study here, the output of the trained models is a qualitative judgment on the type of relationship between two input texts. We will describe some novel and promising ideas on how to use kernel methods to model the classification functions. All these proposed methods will be further detailed in the following chapters.

1.6. Datasets for Semantic Similarity Assessment Evaluation

To evaluate our proposed framework for semantic similarity assessment and all the associated methods for comparing two texts, which will be defined and detailed in the following chapters of this dissertation, we have experimented with three representative datasets, two for paraphrase identification (the MSR and the ULPC corpora) and one for the entailment recognition (the RTE corpus). In the remainder of this section we will describe these three datasets.

1.6.1 The Microsoft Research Paraphrase Corpus (MSR)

Microsoft Research Paraphrase (MSR) Corpus (Dolan, Quirk, and Brockett 2004) is a standard dataset for evaluating approaches to paraphrase identification. Although the corpus has some limitations, which will be pointed out next, it has been so far the largest publicly available annotated paraphrase corpus and has been used in most of the recent studies that addressed the problem of paraphrase identification. The corpus consists of 5801 sentence pairs collected from newswire articles, 3900 of which were labeled as paraphrases by human annotators. The average length for sentences in this corpus is of 22 words. Furthermore, the corpus is divided into a training set (4076 sentences of which 2753, or 67%, are true paraphrases), and a test set (1725 pairs of which 1147, or 66%, are true paraphrases).

There are several critiques about MSR corpus. First, MSR has too much word overlap (spawning form the method used to collect the data set) and less syntactic diversity. Therefore, the corpus cannot be used to learn paraphrase syntactic patterns (Weeds, Weir, and Keller 2005; Zhang and Patrick 2005). Given the high lexical overlap, a good strategy would be to focus on differences among the sentences in a pair. It should be noted that the lexical overlap is recognized by the creators of the corpus (Dolan, Quirk, and Brockett 2004) which indicate a .70 measure of overlap (of an unspecified form). The T-F split in both training and testing is quite similar though (67-33%).

Second, the annotations by humans were made on slightly modified sentences which are different from the original sentences publicly released. For instance, humans were asked to ignore all numbers and simply replace them with a generic token, e.g., MONEY for monetary values, and make judgments accordingly. This discrepancy between what humans used and what systems take as input complicates the task as some decisions are counterintuitive which means someone trying to define a set of meaningful features by inspecting a subset of examples may be puzzled by some of the expert decisions. For instance, the pair below was judged as a paraphrase although the percentages as well as the indices (Standard & Poor versus Nasdaq) are quite different.

> Text A:*The broader Standard & Poors 500 Index .SPX gained 3 points, or 0.39 percent, at 924.*
>
> Text B:*The technology-laced Nasdaq Composite Index < :IXIC > rose 6 points, or 0.41 percent, to 1,498.*

Because of such instances in the corpus, where sentences are intentionally labeled as paraphrases even when the small dissimilarities are extremely important, e.g., different numbers, the evaluation becomes a bit trickier. Here is another example of a pair of sentences from the corpus in which the small difference in both the numbers and the anonymous stocks in Text A are not considered important enough for the annotators to judge the two sentences as non-paraphrases.

Text A: *The stock rose $2.11, or about 11 percent, to close on Friday at $21.51 on the New York Stock Exchange.*

Text B: *PG&E Corp. shares jumped $1.63 or 8 percent to $21.03 on the New York Stock Exchange on Friday.*

This makes the corpus more challenging and the fully-automated solutions look less powerful than they would on a paraphrase corpus that followed the standard interpretation of what a paraphrase is, i.e., the two texts have exactly the same meaning. Nevertheless, the MSR corpus is the largest available and most widely used, and we will also use it in this work as the main dataset for evaluating proposed approaches.

1.6.2 The User Language Paraphrase Corpus (ULPC)

The User Language Paraphrase Corpus (ULPC; (McCarthy and McNamara 2009)) comprises annotations for all three types of relations exemplified in the introductory section of this chapter (paraphrase, entailment and elaboration), as opposed to MSR which only focus on one relation. The corpus was created from students inputs collected during iSTART (Interactive Strategy Training for Active Reading and Thinking) sessions. ISTART (McNamara, Levinstein, and Boonthum 2004; McNamara, Boonthum, and Millis 2007) is an ITS system that provides students with reading strategy training. As part of the training, students are asked to self-explain a given text, called the *textbase*, using a number of reading strategies, such as paraphrasing. The corpus contains 1998 such pairs between a textbase and the students' self-explanations. The pairs are evaluated on 10 dimensions including entailment and paraphrase quality but also on other quality dimensions such as garbage, i.e., incomprehensible input. These other quality dimensions are not text-to-text relations but rather characteristics of a single text, i.e., the writing quality of the student response. An example of a textbase (T) and self-explanation (SE) in iStart that is encoded in the ULPC corpus is provided below (the SE is pasted as typed by the student):

> T: *During vigorous exercise, the heat generated by working muscles can increase total heat production in the body markedly.*
>
> SE: *alot of excercise can make your body warmer.*

The texts in the MSR and RTE data sets, collected from news articles written by professionals, are grammatically correct with almost no spelling errors and, importantly, with many named entities, e.g., Mexico. On the other hand, ULPC texts represent high school students attempts to self-explain textbases. The student paraphrases are less grammatical, with a relatively large number of misspellings, and no named entities. These characteristics of the ULPCs corpus make it special in some sense when compared to the other corpora. For our current task, we decided that we will refrain from dealing with any misspelled or mistyped texts and so we used a modified version of the initial corpus which has been cleaned and corrected by linguistic experts (i.e., terms like *oxygon* or *th* were replaced with their corresponding correct words, *oxygen* and *the*). The average length of the sentences in this corpus is about 21 words. To evaluate our methods we focus on one particular dimension called ”Paraphrase Quality bin”. This dimension measures the paraphrase quality between the target-sentence and the student response on a binary scale, similar to the scale used in MSR. From a total of 1998 pairs about 55% were classified by experts as being paraphrases. For our experiments we will split this corpus in two parts: 1499 instances (75%) will be used for training, and 499 (25%) instances for testing.

1.6.3 The Recognizing Textual Entailment Corpus (RTE)

Textual entailment recognition started as a generic task being proposed by the PASCAL[2] European research group, to measure the semantic textual inference or entailment between texts. As was discussed in the previous chapter, textual entailment differs from textual paraphrasing in being an asymmetric relation from one text, called the entailing "Text" to another text, called the entailed "Hypothesis". The first PASCAL Recognizing Textual Entailment Challenge (RTE-1) took place between Summer of 2004 to Spring of 2005, and it provided the first benchmark for the entailment task. Since then, there had been 5 more RTE Challenge Tasks; first two (RTE-2 and RTE-3) were supported by the PASCAL group still, while the last three (RTE-4, RTE-5, RTE-6) got support from the Text Analysis Conference (TAC) under the National Institute of Standards and Technology (NIST). In our work we will study the relation of entailment on the corpora of first three RTE Challenge tasks: RTE-1 contains 567 pairs of sentences for the training (or development) data, and 800 pairs for the test data; while RTE-2 contains 800 instance pairs for training data and 800 for testing data; and similarly, RTE-3. The average length of the entailed texts in all the three corpora is 31 words, while the average length of the entailed hypotheses is 10 words. Similarly to MSR, the instances are annotated with a binary class, TRUE if there is relation of textual entailment from the first sentence to the second, or FALSE otherwise. The distribution of these classes on all three corpora is even (50%). For our experiments we will use RTE-1, RTE-2 and the training part of RTE-3 as our learning and development corpus, while the testing part of RTE-3 will be used as

[2] PASCAL - Pattern Analysis, Statistical Modeling and Computational Learning

the testing corpus. Therefore, out of all 4657 RTE instances that we use, 3767 (82%) are used for training, while 800 (17%) are used for testing.

1.7. Applications of Semantic Similarity Measures

A well-defined text-to-text semantic similarity metric can have many useful applications. The variety of its applicability depends of course on the particularities of the metric. This section gives some examples where using semantic similarity metrics can help in solving various practical problems and tasks.

The concept of semantic similarity can be understood in many ways, depending on the purpose to which one wants to use it. The most obvious case where such a metric can be used is to analyze whether two given texts tell the same story or not. But that is not always the case. Maybe one text summarizes, infers or concludes some particular facts from the other text. To give a more concrete example, consider a system that analyzes, organizes and summarizes news articles coming from different sources. A useful semantic similarity metric will be one that can tell whether two blocks of text are referring to the same event. They do not necessarily need to relay the same message. Maybe the two articles that enclose these two blocks of text portray conflicting information about the same event. What is of interest here is if the application can somehow recognize that the articles are referring to the same event or topic, and then apply further functions to summarize, combine or compare these two articles.

In Question Answering (QA), multiple answers that are paraphrases of or semantically similar to each other could be considered as evidence for the correctness of the answer (Ibrahim, Katz, and Lin 2003; Rinaldi et al. 2003). For QA systems that use databases of manually generated answers to predefined sets of questions, one can use semantic similarity metrics to determine if a new posed question is a paraphrase to one of the questions stored in the database, for which the answer is known. One of such systems is the Falcon system (Harabagiu et al. 2000), which make use of an ad-hoc module capable of caching answers and detecting question similarity.

In Intelligent Tutoring Systems (Graesser et al. 2005; McNamara, Boonthum, and Millis 2007), the ability to support complex natural language-based dialogue between the virtual tutor and the learner is a crucial component of such systems. In a complex dialogue, the student is encouraged to ask questions and also respond with free natural language answers to the tutor's deep inquiries (e.g., conceptual physics questions) about the student's current level of knowledge. Given these types of free form answers from students, the virtual tutor is expected to validate these answers in one form or the other. A common technique used here is to compare the input with sets of predefined answers that a student could give. These answers may be correct or may be some common misconceptions about the topic referred in the question. The idea is to select the predefined answer that is the most semantically similar with the input and has a similarity degree above a certain threshold. The tutor assumes that this answer is an ideal representation of the input, and will give proper feedback according to the characteristics of the predefined answer: if the predefined answer is correct, the feedback will be positive, if the answer is a misconception, the feedback will be negative but accompanied by constructive explanations in order to repair the discovered learner's misconceptions. A common metric used in Intelligent Tutoring Systems, such as AutoTutor (Graesser et al. 2005) is Latent Semantic Analysis which will be discussed in more detail in Chapter 4.

Table 1.1
Example of ideal and student-generated paragraphs in MetaTutor.

Type	Paragraph
Ideal	The heart is a muscular organ that is responsible for pumping blood throughout the body through the blood vessels. The heart, blood, and blood vessels work together to carry oxygen and nutrients to organs and muscles throughout the body and carry away waste products. The circulatory system works with the system that makes hormones (the endocrine system), which controls our heart rate and other body functions. Blood carries oxygen from the lungs to all the other organs in the body. Metabolism occurs when blood delivers nutrients, such as proteins, fats, and carbohydrates, to our body.
Student	The circulatory system is composed of blood, arteries, veins, capillaries, and the heart. Its purpose is to supply blood flow and oxygen to the body and to pick up waste (carbon dioxide). Blood is either oxygen rich or poor. Oxygen poor blood needs to return to the lungs from the heart to get more oxygen. Once blood has generated through the body it's oxygen is depleted. Needs to get back to the heart so it can get back to lungs.

In a related task, automatic detection of student mental models in MetaTutor (Azevedo et al. 2008), an intelligent tutoring system that teaches students self-regulatory skills, a challenging task is deciding how similar a student-generated paragraph is to an ideal, expert-generated paragraph. The student-generated paragraphs are obtained from the prior knowledge activation (PKA) meta-cognitive activity in MetaTutor when students are prompted to write a paragraph outlining everything they know about a given learning goal, e.g., learn about the human circulatory system. Table 1.1 shows an example of an ideal paragraph compared to a student-generated paragraph. In this case, the task is to assess how semantically similar the two given paragraphs are.

Semantic similarity can also be useful to help in the detection of plagiarism. Given a paper written by a certain researcher named Bill and a set of articles published by other researchers but closely related to the topic presented by Bill, the goal is to measure how much of Bill's work is original and how much is ”inspired” from other people's work. Obviously we do not expect to have a system that will be able to solve the whole plagiarism detection task, but having a properly defined similarity metric could assist and warn the raters of Bill's paper, for potential plagiarisms issues to other particularly specified articles.

A recent recommendation to the field of Software Testing is to use NLP-based techniques in detecting duplicate bug reports (Rus et al. 2010). Defect reports are detailed descriptions, made in natural language format, for problems detected in a software product. The reports are stored in a defect database. As it turns out, not all defect reports in this database are unique because often the same problem is discovered by different testers and therefore reported independently. Reports that describe the same underlying problem are called duplicate reports, or dupes. Duplicates are numerous in bug databases and automated tools to automatically assess the semantic similarity of defect reports are highly desirable, to help decrease the time these defects are analyzed and then properly fixed.

Text-to-text semantic similarity metrics can also be very helpful for clustering texts that are similar in meaning. Clustering texts is an important task for many applications in data mining and information retrieval, such as intelligent retrieval of textual data that is relevant to a given topic of interest. Much research has been done in data mining for text clustering, some using bag-of-words approach (Baeza-Yates and Ribeiro-Neto 1999), other using more complex approaches using ontologies of words (Jing et al. 2006) extracted from well-known lexical databases, such as WordNet (Miller 1995). Through our proposed framework, we hope to help and expand the research in this area by offering a wide range of text-to-text semantic similarity metrics, which can be used as distance measures in various methods of text clustering.

1.8. Contributions

There are several contributions put forward by the current work. First, we make an attempt to formalize the problem of semantic similarity and present it in a broader perspective. Second, according to this formalism, we propose a framework upon which various methods can be developed in order to solve particular problems of semantic similarity. Third, several methods are proposed and validated through experiments.

A lot of research has been done on this topic, particularly on two problems of textual semantic similarity: paraphrase identification and textual entailment. As previously mentioned in the introductory section of this chapter, the work of Androutsopoulos and Malakasiotis (Androutsopoulos and Malakasiotis 2010) provides a good overview on these two problems, although not complete, since new related work is published every year. Some researchers have worked on paraphrase identification (Zhang and Patrick 2005; Fernando and Stevenson 2008; Rus et al. 2008a; Das and Smith 2009), some on textual entailment (Dagan, Glickman, and Magnini 2005; Rus et al. 2008b), while others searched for generic solutions to solve both problems (Corley and Mihalcea 2005; Finch, Hwang, and Sumita 2005; Wu 2005; Zhang and Patrick 2005; Qiu, Kan, and Chua 2006; Ramage, Rafferty, and Manning 2009; Heilman and Smith 2010). There is also a lot of variety and ingenuity in the methods that were proposed. In regard to corpus-based training, some are fully supervised (Zhang and Patrick 2005; Qiu, Kan, and Chua 2006), while some are only partially supervised (Wu 2005; Rus et al. 2008b). Some methods rely on word-based semantic similarity measures

(Corley and Mihalcea 2005; Fernando and Stevenson 2008), some on syntactic information, such as dependencies (Wan et al. 2006; Heilman and Smith 2010), or verb predicate forms (Qiu, Kan, and Chua 2006). Some approaches rely heavily on classifiers (mostly SVMs) to learn from features based on machine translation evaluation techniques (Finch, Hwang, and Sumita 2005), quasi-synchronous dependency grammars (Das and Smith 2009), or surface string similarity with synonyms and dependency overlap (Malakasiotis 2009).

A few researchers, including the author, argue that the nature of the available corpora for the task of semantic similarity analysis, i.e., MSR (Dolan, Quirk, and Brockett 2004), is debatable in terms of how representative their instances are for particular tasks of textual semantic similarity (Weeds, Weir, and Keller 2005; Zhang and Patrick 2005). For this reason, we believe it is a rather difficult task to declare one clear winner from the different approaches proposed, based purely on the performance numbers. One could argue that, to a certain extent, each one of these approaches is valid enough, if the theoretical reasoning upon which they are based is properly motivated from a linguistic point of view, and their reported performance scores are at least competitive. This dissertation advances the idea of a novel approach based on a general framework to cover a wide spectrum of textual semantic similarity problems and with a good theoretical foundation that will make it fairly acceptable in the research community. We hope this framework will offer an environment in which various levels of linguistic information can be used to provide a more rigorous comparison between different approaches, which rely on various combinations of the available linguistic information. Based on this framework, we propose, explore, and validate through rigorous testing a suite of methods to the task of semantic similarity, with various degrees of sophistication. The most promising methods proposed are found to offer competitive results with other state-of-the-art solutions. We validate our methods across two of the most studied relations of semantic similarity, paraphrase and entailment.

As previously noted, previous attempts to address both problems of paraphrasing and textual entailment recognition have been reported before, but no one, to the best of our knowledge, have addressed the problem of semantic similarity between texts with a general framework that permits the development of a suite of methods which vary in their degree of sophistication. Heilman and Smith (2010) describe Tree Edit Models, which represent sequences of tree transformations to model pairs of semantically related sentences. These models are presented as viable approaches to the tasks of recognizing entailment, paraphrase identification, and answer selection for question answering. Ramage, Rafferty and Manning (2009) suggest using local relatedness information extracted from *random walks* over graphs constructed from underlying to compute the text semantic similarity. They tested this approach on paraphrase and entailment relations at sentence level. On the same tasks, Finch, Hwang, and Sumita (2005) propose using machine translation evaluation scores (such as BLEU, NIST, WER or PER) to measure semantic equivalence at sentence level. In another paper, McCarthy et al. (2008) analyze lexico-syntactic approaches for the textual relations of entailment, paraphrase and, elaboration, which are hand-coded into a corpus of self-explanations taken from iStart, an Intelligent Tutoring System (McNamara, Boonthum, and Millis 2007).

This current work makes an attempt to expand much of the previous work done, by proposing an overarching, flexible framework for computing text-to-text similarity that ranges from simple overlap to statistical semantic similarity, i.e., LSA, to similarity in highly-dimensional spaces, i.e., SVM. In addition, we present a systematic study of the different levels at which one can compute semantic similarity between two given, short lengthened texts.

In summary, this dissertation proposes and studies several methods and metrics to compute various semantic similarity values, in order to solve entailment and paraphrase recognition problems. Evaluations of these methods on common data test sets are presented. We start with a suite of simple word-based methods to measure semantic similarity between two sentences. These methods are basic extensions of some word-to-word similarity metrics (WordNet similarity) and, although they are simple methods, they behave surprisingly well on the available data sets. We then introduce dependency relations and explore how they can be used to measure text-to-text similarity. A somewhat different approach using vector-representations of texts, based on Latent Semantic Analysis (LSA), is also investigated. We contribute to the LSA related research by analyzing the effects of using various weighting schemes to compute similarity scores between texts. One particularly interesting weighting scheme that we will experiment with LSA is the inverse document frequency scheme computed from Wikipedia, currently the largest online and freely available collection of documents. Furthermore, we present a novel and ambitious approach of using word-based and dependency-based kernels to measure and classify semantic similarities. Our experiments with these kernel-based methods show promising and competitive results when compared to other methods.

1.9. A Look Ahead

The current proposal is organized in six major parts and a concluding section. Thus far, the introductory chapter presented the research problem of measuring the semantic similarity between texts, and outlined the main ideas being proposed by this work and its contributions. A general framework is then described, for solving the current research problem, along with three main datasets, which will be used to evaluate the proposed approach. In the end, this first chapter listed several applications or tasks where such a framework could become useful. Chapter 2 further details the research problem with some more representative examples. Next, the meaning or data representation proposed in Chapter 1 is described in finer lines. This second chapter then talks about how to automatically derive such representations from input texts given in raw form, as part of a preprocessing phase.

The next four chapters propose and study several methods and metrics, to measure the semantic similarity between texts, in order to solve the particular cases of entailment and paraphrase recognition problems. Evaluations of these methods on standard datasets are also presented at the end of each chapter. Chapter 3 presents the lexical methods, which are based on computing the lexical overlap between two input texts, in order to measure the semantic similarity between them. We extend these methods with various weighting schemes for each lexical token, which can be computed either locally, from the current input texts, or globally, from a much larger collections of texts. One particular type of global weight that we use is based on the inverse document frequency (IDF) values. The process for computing these values from Wikipedia, the largest collection of online documents on general knowledge facts available at this time, is fully detailed in Section 3.9. On the next fourth chapter, we will further extend the lexical methods with some word-to-word semantics. We study two types of such metrics computed at the word level, one based on the study of manually defined relations between semantic concepts that are being stored in WordNet - a lexical database and taxonomy of concepts, while the other is based on latent semantic analysis (LSA), a vectorial representation for the meaning of words, which became recently popular and was successfully used in some Intelligent Tutoring Systems (Graesser et al. 2005; Dessus 2009). We also present in this chapter how to apply some smart word-based weighting schemes when calculating the semantic similarity metrics between words. We contribute to the LSA related research by analyzing the effects of using various weighting schemes to compute similarity scores between texts. Chapter 5 introduces dependency relations and how they can be used to measure text-to-text similarity. Chapter 6 presents a novel and ambitious approach of using word-based and dependency-based kernels to measure and classify semantic similarities.

The dissertation ends with a concluding chapter where, along with some concluding remarks and future directions on the work being presented here, it also describes in more detail some of the previous work done by other researchers on the task of semantic similarity assessment, and will further compare their evaluation results with the best ones obtained by our proposed research methods.

CHAPTER 2

THE MEANING REPRESENTATION

2.1. Introduction

In the introductory chapter, we briefly described the task of measuring semantic similarity between texts and its applications; we defined what we understand by text-to-text semantic similarity and how we intend to measure it. We also suggested a three phase approach to assessing similarity which starts with the input texts being mapped onto a semantic representation where lexical, syntactic, and semantic information is retained. The resulted representations can then be used in the subsequent phase in which second-order features are being extracted and which reflect various similarities and even dissimilarities between the target input texts. In the last phase, these features are used to compute a semantic similarity score between the two input texts.

This chapter elaborates on the first phase of our approach, the semantic representation (which was briefly introduced in Section 1.5.1). It details the representation of texts and the process of automatically deriving the representation, through a semantic extraction process where texts are analyzed at lexical, syntactic, and semantic levels. We begin with a careful analysis of what it means to quantify the semantic similarity between texts.

2.2. A Closer Look to Text-To-Text Semantic Similarity

This section presents several illustrative examples with respect to assessing similarity between two texts. The ideas of computing semantic similarity that emerge from these examples motivate the choices we have made to develop our meaning representation which is to be used later when defining and computing the sentence-level semantic similarity metrics.

In the simplest case, two input texts can be considered semantically equivalent when there is an almost perfect *overlap* among the lexical tokens in the two texts and the syntactic relations among these tokens are also similar. By *overlap* we mean that the tokens are represented by their sense and not their lexical form. In other words, we are comparing the *canonical forms* of the two texts, where: a) all lexical terms are replaced with the identifier of the semantic sense to which they belong (e.g., the ID number of a WordNet synset), and b) all syntactic relations are converted into a agreed-upon format (e.g., use only active voice for verb phrases). If the canonical forms are identical or equivalent, then we say that the texts are also equivalent and therefore convey the same meaning. Such ideal cases rarely occur in the real world.

The next level of complexity occurs when two sentences are almost identical with the exception of a few concepts which are present in one text but not the other. Depending on the semantic function of the extra concepts, they may have a smaller or a greater significance on the overall meaning of the individual sentence in which they occur and therefore on the similarity of the two sentences. For example, consider the next group of sentences:

A. *John ate an apple.*

B. *John ate a red apple.*

C. *John ate a green apple.*

D. *John bought an apple.*

Sentence A and sentence B are almost identical except the extra attribute of color (e.g., *red*) in sentence B. As *red* is just a modifier, the two sentences could be considered fairly similar. The same holds for sentences A and C. However, we cannot say the same thing about sentences B and C, since we know in this case that the color of the apple is different. A similar line of reasoning can be applied when comparing the two semantically different sentences A and D, in which the predicate of the two is different.

Sometimes, sentences refer to the same concepts by using different words, i.e., synonyms, hyponyms, or anaphors. Automated methods assessing similarity of texts should account for such cases. Suppose now that we want to compare sentence A above with the following two sentences:

E. *John consumed an apple.*

F. *John ate a fruit.*

In sentence E, the verb ”to consume” is a synonym of the verb ”to eat”, while sentence F contains the word ”fruit” which is a hypernym of the word ”apple” (i.e., an apple is a fruit). Knowing about the semantic relation between two words can help us to correctly find the relation of paraphrase between sentences A and E and the relation of entailment from sentence A to sentence F. In a looser definition of paraphrase in which, say, we are only interested to know that John ate some type of fruit, then we can safely consider sentences E and F as being semantically similar too.

Both examples above can be solved by considering the similarity of the few mismatching words, given that everything else matches. Such cases can be handled by using libraries that offer explicit semantic relations between words, such as synonymy or hyponymy. To get word-based semantic similarity metrics, we rely on two approaches. One is using the WordNet Similarity package (Pedersen, Patwardhan, and Michelizzi 2004) which offers six measures of similarity and three measures of relatedness, based on the lexical database WordNet. The second approach is using Latent Semantic Analysis (LSA), a way of representing the meaning of words with vectors, where the similarity between two words can be computed as the normalized dot-product between their corresponding vectors (Landauer et al. 2007).

Syntactic relations in a sentence are also important to assess its meaning as the following example shows. Suppose we need to assess the semantic similarity of the following pair of sentences:

A. *Yahoo bought Overture.*

B. *Overture was bought by Yahoo.*

This is a very simple example of paraphrase where the main difference is the presence of active tense for the predicate in sentence A versus the passive tense in sentence B. By looking at the dependency relations in the two sentences, we notice that the logical subject, Yahoo, and the direct object, Overture, is the same in both sentences, which suggests that the two sentences are in fact paraphrases. Dependency relations can also tell us what words are important or not for a sentence. For example, a word that describes an attribute of a noun, such as the color of a fruit, is considered less important than other types of words, such as the subject or the predicate of a sentence.

Another example where syntactic information can help is when one sentence contains some extra information that can be ignored. Suppose we have the following pair of sentences:

A. *Tommy said that John proposed to Mary.*

B. *John proposed to Mary.*

Sentence A contains some extra information when compared to sentence B: ”Tommy said that ...” Sometimes we are only interested in the real message contained within the sentence, which is the fact that John proposed to Mary, and we can ignore the rest. The syntactic information will tell us that there is an extra phrase which complements the main phrase and can be ignored. This example was inspired from the work of Qiu, Kan and Qua (2006) where a decision tree classifier is trained to find what bits of extra information should be considered significant or not, while checking for paraphrases.

The syntactic relations of words in a sentence can be represented either with phrase-based syntactic trees or with dependency trees. Phase-based trees are hierarchical, tree-like structures, where the leaf nodes represent words, in the same order as they appear in the text, from left to right, while the intermediate nodes represent parts-of-speech or larger phrases (e.g., noun phrases - NP, verb phrases - VP, sentence - S). Figure 2.1 shows an example of a syntactic tree, based on Penn Treebank annotation guidelines (Bies et al. 1995). Dependency trees are hierarchical organizations of explicit syntactic dependency relations between two words in a sentence. In a dependency tree, every word in the sentence is represented by one node in the tree and modifies (or *is modifier of*) exactly one word, its head, except the head word of the sentence, which does not have a head and is the root node of the tree. Figure 2.2 depicts a dependency tree extracted using Minipar (Lin 1993), a dependency parser. The nodes in the tree are labeled with the lemma form of the initial words. The types of the dependency relations between words are represented as labels on the edges of the tree. Note that there is no straight and simple way to represent the initial sentence directly from the dependency tree, as is the case with the phrase-based trees, where the sentence could be reconstructed by reading the leaf nodes from left to right.

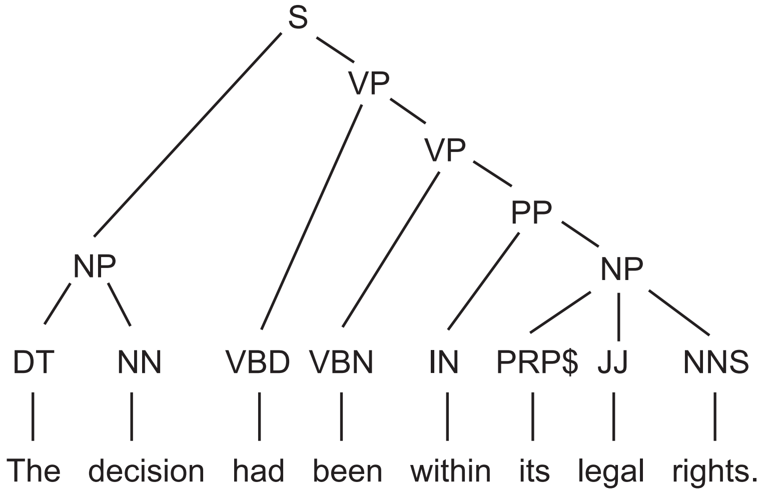


Figure 2.1
Example of a Phrase-based Syntactic Tree

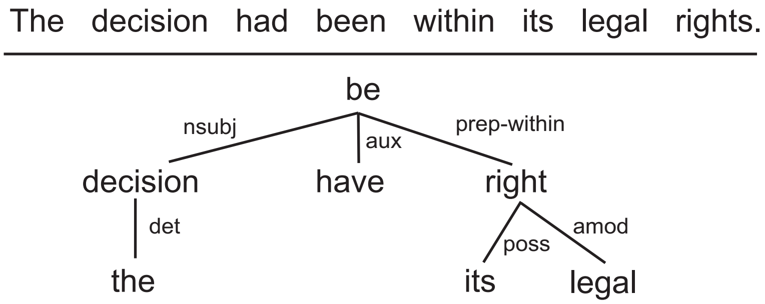


Figure 2.2
Example of a Dependency Tree

2.3. Semantic Interpretation versus Semantic Mapping

Much research has been conducted on quantifying and measuring text-to-text semantic similarity. The ideal approach to handle this problem is a complete semantic understanding of the input, which we call the *semantic interpretation approach*. With current technological resources this approach is not yet practically feasible. However, significant progress has been made with the development of linguistic and semantic-based corpuses such as WordNet (Miller 1995), PropBank (Palmer, Kingsbury, and Gildea 2005), and FrameNet (Ruppenhofer et al. 2005).

Because the ideal approach is not yet practically feasible, scientists have tried going more towards a *semantic mapping approach*. The goal in semantic mapping is to assess the meaning of one text relative to the known meaning of another. For instance, in Intelligent Tutoring Systems finding the correctness of natural language student input can be reduced to assessing the meaning with respect to an ideal answer provided by an expert whose correctness is assumed. Instead of truly understanding student input's meaning, all we need to do is finding a good mapping function through which we can accurately measure the semantic similarity between the student input and the benchmark text, the exert answer in this example. In order to facilitate the semantic mapping approach, the texts to be assessed are converted in a meaning representation that retains the prominent linguistic information of the original texts. These representations are then compared using mathematical and algorithmic functions and similarity scores are computed. This chapter describes mapping the source texts onto such representations. We describe the representation in detail and show how to automatically map texts onto it.

2.4. The Semantic Representation

In this section we detail the design of the meaning representation. There are two requirements that we wish to fulfill through this representation. First, the representation should be *computer-friendly*, well-structured, rigorous and unambiguous, for an easy and straightforward automatic knowledge processing. Second, the representation should be ergonomic and easy to read, understood and analyzed by humans; because a representation that is easy to understand, makes a good starting point to develop and debug new research ideas. Our proposed representation is based on a class of knowledge representations, called *natural language based knowledge representations* (NL-based KRs), which meet both of the above requirements.

Given the input text T of relative short length (it can be a phrase, a sentence or a paragraph) we first represent it as an ordered list of lexical tokens. A token can be either a word or a punctuation mark and it is not unique (there can be more than one token with same lexical form). We associate a representational element to each token (*tok*) which is described in the following regular expression form:

$$SR_{tok} = (\, Word, Lemma, POS, Specificity, \text{WN-SENSE}|\text{LSA-Vector}, \\ (< - : dep_{type} : dep_{mod} > | < dep_{head} : dep_{type} : - >)+)$$

If the token corresponds to a punctuation mark, then only the first three items are considered. *Word* and *Lemma* are identical in this case and are assigned a label that indicates the corresponding punctuation mark (e.g., *dot, comma, colon*). The *POS* is allocated the constant string value *"PUNC"* which specifies that the current token is associated to a punctuation mark in the input text, or *"PERIOD"*, which specifies the end of a sentence.

For tokens that correspond to actual words in the input text, the items of the representation are described below:

- *Word* represents the raw, unmodified lexical form of the word. By storing it this way we make sure nothing is lost from the initial input, which we can easily reconstruct by concatenating all the *Word* elements of the list.

- *Lemma* is the lemma, or the root form, of the initial word, in lower case letters. This property will allow us to group words that are similar in meaning and differ only by their lexical derivations of the same root form (e.g., *pretty* is the root form of *prettily*). Words that have same root form or lemma are considered to be semantically close. Another option that we can use instead of lemmas is the *stemmed* form of words. Stemming is a simple process for reducing the morphologically derived words to their base (root) form. The advantage for *stemming* is that it is easy to implement and is fast. The disadvantages are that not all stems are dictionary words (i.e., the stem of *prettily* is *prettili*, based on Porter Stemmer's rules (Porter 1997)), some words with same lemma have different stems (i.e., the stem form of *leaves* is *leav* and not *leaf*), and some words that have the same stem don't always represent for the same concept (i.e., both words *generic* and *generate* have the same stem, *gener*).

- *POS* is the part-Of-speech, or the syntactic role of the word (e.g., noun, verb, adjective). This item is important when we want to study and compare the syntactic functions of words. For example, determiners (e.g., *a*, *an* or *the*) are less important, and sometimes can even be ignored, when determining the semantic similarities between two texts (e.g., *"I saw a boy run."* vs. *"I saw the boy run."*).

- *Specificity* measures the specificity of a word. By definition, the specificity of a word tells us how common the word is in general language use. Earlier studies (Corley and Mihalcea 2005) suggest that using specificity-based weighting for words can improve a system's performance on the task of paraphrase identification. If a word is very specific it is deemed more important for the task of similarity assessment since its usage will have a greater influence on the meaning and the topic of the larger text in which it occurs.

- Next item represents the semantic meaning of a word. We adopt two possible options to represent the meaning of a word. First one is the sense of the word from Wordnet (Miller 1995), one of the most commonly used and largest lexical databases for English. Second option is Latent Semantic Analysis (LSA), a well-known technique for capturing the meaning of words through vectorial representations. LSA (Landauer et al. 2007) suggests that the meaning of a word is characterized by the company it keeps. By company we understand the context in which the word is commonly used. This context can be evaluated by analyzing word-to-word co-occurrences in large collections of texts. The co-occurrences of a word with other words are synthesized in its associated LSA vector. Since LSA vectors are rather tricky to represent ”on paper”, we list only WordNet's sense numbers in all following examples of meaning representation. More details about WordNet senses and LSA vectors will be presented in the following subsections.

- Last item in our representation is a list of word-to-word syntactic dependencies which are associated with the current word. The syntactic information for an individual word in a sentence is extracted from a dependency tree. Given a dependency tree, the list of dependencies can be easily derived by traversing the tree. We characterize the dependency by the *type* of the relation, the *head*, and the *modifier*. When the current word is the head of a dependency, then we represent the dependency in our list as $< - : dep_{type} : dep_{mod} >$, where dep_{type} is the type of the dependency and dep_{mod} is the modifier in the dependency relation. In the other case, when the current word is the modifier in the dependency, then we represent the dependency as $< dep_{head} : dep_{type} : - >$ where dep_{head} is the head word in the dependency. Notice that we use a dash line as a replacement for the current word. We represent the dependencies in a way that allows for easier comparisons between the dependencies sets of two words.

The role of this representation is to provide easy access to any type of information that we may inquire. Our experiments have shown that choosing the right type of information from this representation can significantly affect the performance of the system, whether we want to include punctuation into the next steps of the analysis, or we choose to ignore the case-sensitive aspect on the input words.

Some of the items presented in the list above require more details, which are further discussed in the next subsections.

2.4.1 Word Senses in WordNet

Most of words in English are polysemous; they have multiple meanings depending of the context in which they are used. WordNet (Miller 1995) is organized in *synsets*, which are sets of synonymous words referring to a common semantic concept. The sense of a word is associated with exactly one synset.

Example of such synsets are (*"reason, understanding, intellect"*) defined in WordNet as: *the capacity for rational thought or inference or discrimination*; or (*"cause, reason, grounds"*) defined as: *a justification for something existing or happening*.

The WordNet database maintains definition entries on four part-of-speech types that associate with content words only: nouns, verbs, adjectives and adverbs. Some words can be used with more than one part of speech. Therefore the senses of a word are also associated with a particular part of speech. The senses are numbered within the definition of a word and the first sense is known as the most used sense for the word and its part-of-speech. As an example, let's look at the word *reason* as is stored in WordNet 2.1. The word has two part-of-speech types: a noun, which has 6 senses (1. *"rational motive"*, 2. *"an explanation"*, 3. *"the capacity for rational thought"*, 4. *"good sense and sound judgment"*, 5. *"a justification"*, 6. *"a fact that logically justifies some premise"*); and a verb, which has 3 senses (1. *"to decide by reasoning"*, 2. *"to present arguments"*, 3. *"to think logically"*). If we want to use the word *reason* with the meaning for *"a justification"* then we refer to is as *the fifth sense of the noun*.

Knowing the sense of the words used in texts is a big advantage for any task that requires a semantic understanding. However, finding the correct sense of a word in a given context, e.g., a sentence, is a major challenge in NLP, known as the problem of word sense disambiguation (Jurafsky and Martin 2002). The difficulty of the problem arises from two reasons. First, detecting the correct sense of a polysemous word requires deep knowledge of the language and the ability to analyze the text that describes the context in which the word is used. Second, we might encounter new words or new senses of already known words that are not present in WordNet's dictionary.

2.4.2 Using LSA to represent the meaning of words

LSA (Landauer et al. 2007) is a statistical technique for representing meaning of words that relies on word co-occurrences to derive a vectorial representation for each word. It is based on the principle that the meaning of a word is defined by the company it keeps. Two words have related meaning if they co-occur in the same contexts. The co-occurrence information is derived from large collections of text documents. The mathematical details of the process for constructing the LSA vectors of words are presented below.

In a first step, a term-by-document matrix X is created in which element (i, j) contains a binary value, 1 if term i occurs in document j and 0 otherwise. We can also use weighted values, which should reflect the importance of the terms for the document where they appear. For example, we can use the frequency of how many times term i appears in document j. The importance of a term can be determined either locally, based on its appearance in the current document, or globally, based on its usage in general. Normally, we would associate two weights to the elements of the matrix X, a *local weight* and a *global weight*. The most commonly used local-global weighting scheme in this case is a global entropy with a local log-type frequency (more details about these weights are given in Sections 3.4 and 4.8.1). After the term-by-document matrix is created, a mathematical procedure, called Singular Value Decomposition (SVD), is applied resulting in three new matrices: T and D, which are orthonormal, and S, which is a diagonal matrix, such that $X = TSD^t$. The dimensionality of these matrices is then reduced by retaining k rows and columns corresponding to the highest k values in S. A new matrix $X' = T'X'D'^t$ can now be computed that is an approximation of the original term-by-document matrix X in a reduced space of k dimensions. Usually, k takes values between 300 and 500. Every word in the initial collection of documents is characterized by a row (or vector) in the reduced matrix X′. These vectors supposedly characterize the words using so-called latent concepts, one for each of the k dimensions of the reduced space.

The main advantage is that by replacing the WordNet sense of a word with its corresponding LSA vector the hard task of word sense disambiguation is avoided since the meaning of a word is already captured in the LSA vector. Furthermore, in this representation we could capture meanings of words in particular domains, e.g., conceptual physics domain, by simply creating an LSA space based on a collection of texts from that domain. A domain specific LSA space can be created automatically (helping scalability) as opposed to a WordNet-like solution of word sense inventory creation which is mostly manual.

The disadvantage of using LSA is that in an LSA space there is only one LSA vector for a word. Therefore, words that are used with multiple senses in the collection of documents from which the LSA space is being built will have the meaning of all their senses assimilated and mixed in the final LSA vector. This is one unfavorable aspect of the LSA on which most researchers are willing to compromise due to the simplicity of the process of LSA information extraction.

2.4.3 Syntactic Dependency Trees versus Phrase-based Trees

There are two most common ways in which syntactic information can be encoded, either through dependency trees or through phrase-based trees. An important question that we need to answer is why we chose dependency relations instead of phrase-based structures to represent the syntactic information. Besides being a matter of preference, dependency relations can handle free word order languages, such as Hungarian, Portuguese or Russian (Covington 1990). Phrase based trees are heavily dependable on the order of the words in a sentence, and therefore cannot handle free word order languages too well (Covington 1990). By choosing dependency relations we insure the applicability of our method to these languages as well. Another reason for choosing dependencies is that they generalize better the syntactic formations and is easier to compare them between texts.

2.5. Extracting the Semantic Representation

This section will detail the steps to convert the input raw texts into the data representation described in the previous section.

2.5.1 Preprocessing: Lexical Analysis

An important step when working with raw input text in any NLP related task is the preprocessing phase. By preprocessing we understand a set of simple operations that are already proven to be efficient, fast and with high accuracy on the output. A major part in preprocessing a raw input text relates to lexical analysis, which is the lowest and easiest level to analyze a natural language text.

The first step in lexical analysis of the input text is tokenization. Tokenization is the process to obtain the ordered set of lexical tokens that make the text. These lexical tokens can be words, punctuation marks, symbols or other meaningful structures. In general tokenization is a simple process and it can be done easily by most of the NLP libraries available on the internet. Some of the basic operations include separating the punctuation from the words (e.g., adding a space between the comma or period signs and their previous words) or separating some complex words in individual tokens (e.g., separating *"don't"* to *"do n't"* which means a negation of the word *"to do"*). Although a simple process, tokenization is a very important process in NLP related tasks and should not always be taken for granted, especially when working with texts from other languages, such as Thai or Chinese. As Webster and Kit (1992) were saying ”it is an obvious truth, however, that without these basic units clearly segregated, it is impossible to carry out any analysis or generation”.

Besides building the list of lexical tokens, the other role of tokenization is to divide the input text into sentences. Tokens that are labeled as a *period* marks, divide the output list into sentences. This process is also known as the task of sentence detection. We previously noted that the types of any non-word tokens, are stored in the POS field of our data structure.

After obtaining the list of lexical tokens, next steps are POS tagging and Lemmatization. *POS tagging* is the process of labeling each lexical token with a syntactic type. *Lemmatization* is the process for extracting lemmas from words. The order on which one of these two processes is done first depends on the methods that are being used. Sometimes a Lemmatizer requires knowing the POS of a word, and sometimes it does not. State of the art lemmatizers and POS taggers are available and their performance is significantly high. For example, on French language, their performance has been reported as up to 97.68%, for POS tagging, and up to 98.36%, for lemmatization (Seddah et al. 2010).

2.5.2 Extracting word-based semantic information

The next phase in building our semantic representation is to represent individual words based on their semantic meaning in the context of the text. We have mentioned before that there are two ways to represent the meanings of words. One is by using the sense of the word as is stored in a lexical database or dictionary such as WordNet. To find the sense of the word in a specified context is a very hard task if close to perfect accuracy is needed. However, because almost all words have one sense in which they are used most of the time, a simple baseline method of always choosing the most common sense of a word usually gives very high accuracy scores, and this will suffice on most NLP related tasks. Since in the current work, the interest is not to solve the word sense disambiguation problem, we chose to simply associate words with their most used semantic sense.

The second way to represent the meaning of a word is by its associated LSA vector. LSA vectors are included in an LSA space which is previously computed from large collections of documents. Depending on the particularities of the input texts that are given for the task of measuring semantic similarity, we should make sure that the LSA space which we use conforms to these particularities. That is, the meaning of words being used in the input texts needs to be the same with the meaning of these words in the document collection from which the LSA space was built. One straightforward way to make sure that this condition is satisfied, is to build the LSA from a collection of documents that appertains to the same topic as the input instances.

2.5.3 Extracting Dependency Relations

The final step in building our data representation is to retrieve all syntactic dependencies from the input texts and build the list of associated dependencies for every word. We are going to experiment with two dependency parsers: Minipar (Lin 1993) and the Stanford parser (de Marneffe, MacCartney, and Manning 1993). The parsers take as input a tokenized sentence and return as output a dependency tree from which the list of dependencies can be easily extracted. By traversing this tree, for each internal node, which we know is head of at least one dependency, we retrieve triplets of the form *rel(head, modifier)* where *rel* represents the type of dependency that links the node, i.e., the head, to one of its children, the modifier. Although our representation stores only the list of dependencies, it will also be useful to know the position of these dependencies in the tree. As the work of Wan et al. (Wan et al. 2006) suggests, dependencies should also be accounted for their importance in the sentence, just like words. For example a dependency between the main verb and the subject of a sentence ought to be more important than a dependency between a noun and its determiner. Therefore, besides the type of the dependency, we argue that dependencies which are closer to the root in the dependency tree are more important than the others, when measuring for the semantic similarity.

Because in general, dependency parsers cannot produce perfect output, methods that use this syntactic information to measure semantic similarity will have their performance affected by the performance of the parsers. However, since dependency parsers are consistent in their mistakes, these mistakes should be similarly propagated to both texts that are to be compared. In such cases, one would hope that the semantic similarity measures will not be affected too much because the result for comparing two similarly incorrect syntactic constructions will be the same as when comparing two similarly correct syntactic constructions.

In Chapter 5 we describe methods that will use these dependency relations to help in measuring the semantic similarity between texts, particularly on the task of paraphrase identification.

2.6. Experimental Setup on Extracting the Semantic Representation

The main topic of this work is to research models, methods and metrics to quantify the notion of semantic similarity between texts. So far, in this second chapter, we have presented a model of how to process and represent the input texts in a task of measuring the semantic similarity. We also described how to automatically derive these representations using various NLP techniques such as tokenization, lemmatization, POS tagging and extraction of syntactic dependencies. The following chapters will present various approaches to use these representations in solving two particular tasks, paraphrase identification and entailment recognition. For these tasks we will experiment with three representative datasets, which were previously introduced in the introductory chapter. For the remainder of this chapter we will describe our own implementation for preprocessing these datasets and extracting the semantic representations. We also report for each dataset time-related performance scores of the preprocessing phase. In particular, we note that the parsing of texts and the extraction of the dependency trees are the most time consuming steps in this preprocessing phase.

To preprocess the datasets we use two common NLP libraries: the OpenNLP JAVA library [1], which offers standard NLP functions for text processing, including POS tagging and syntactic phrase-based parsing but not dependency-based parsing; and Stanford CoreNLP (StanfordNLP), a similar integrated suite of NLP tools for English in Java, which offers additional support for dependency based parsing, named entity recognition and coreference resolution. Since OpenNLP does not have functionality for dependency-based parsing, we will use the Minipar dependency parser to run some of our dependency-based experiments. Except for the Minipar parser, which was implemented in C++, all our

[1] OpenNLP is currently offered as a free java-based library by the Apache Software Foundation at http://incubator.apache.org/opennlp/index.html

experiments are implemented in Java and are run on a Windows-7, 64k-bit machine, with an Intel i7 CPU @2.67GHz and 9GB of DDR3 memory @1333MHz.

The Microsoft Research Paraphrase Corpus (MSR). To lexically preprocess this corpus (tokenize, sentence detection, lemmatize and POS tagging) using OpenNLP (no dependency parsing), it takes about 29 seconds; using the Stanford Parser (excluding dependencies) it takes 18 seconds. For dependency parsing, with Minipar it takes 69 seconds to parse the whole corpus, while with the Stanford parser it takes 35 minutes and 43 seconds to parse and extract the dependencies. We see that there is a huge gap in time performance between the Minipar and the Stanford Parser and we should explain that difference next. To build the dependency tree, the Stanford parser first needs to construct the phrase-based syntactic tree and then, from this tree it will extract the syntactic dependencies. In opposition, Minipar does not need to use phrase-based parsing for its dependencies. Minipar was developed in C++ which results in execution code that is much faster than Java based programs, and aims precisely at extracting the dependency relations and nothing else. Since most of our work was done in Java, we used Minipar to experiment only with the MSR corpus, and therefore we do not report performance scores using Minipar on the other two corpora.

The User Language Paraphrase Corpus (ULPC). To lexically preprocess the ULPC corpus, with OpenNLP (no dependencies) it takes 9 seconds, while with StanfordNLP (no dependencies) it takes 8 seconds. For parsing dependencies with the Stanford parser, it takes a total of 7 minutes and 23 seconds.

The Recognizing Textual Entailment Corpus (RTE). To lexically preprocess our RTE combined corpus, with OpenNLP (no dependencies) it takes 23 seconds, while with StanfordNLP (no dependencies) it takes 17 seconds. For parsing dependencies with the Stanford parser, it takes a total of 33 minutes and 47 seconds.

CHAPTER 3

LEXICAL-BASED SIMILARITY METHODS

3.1. Introduction

Previous chapters of this dissertation have introduced the reader to the task of measuring text-to-text semantic similarity and how to preprocess and convert the initial input texts into some well defined semantic representations that will help subsequent steps of the measurement process. We refer to these steps as methods or metrics for measuring text-to-text semantic similarity. In this chapter we begin to look at some basic, simple and intuitive methods that use the semantic representation defined in previous chapters. The methods presented in this chapter primarily rely on finding matches between individual words or tokens contained in the texts and using the matched pairs to analyze the input instances at surface string (i.e., lexical) levels only. Some of these methods are simple enough that they can be used as baselines for the evaluation of other, more complex approaches, which we will present in the following chapters.

We further augment the presented token-based matching methods with weighting schemes as follows. For every token in the input texts we associate a weighting value that is statistically computed based on the token's local and global use, in the current texts and in language in general. Local weighting is computed based on token's frequency in the current text, while global weighting reflects the global specificity of the token. The term specificity was briefly mentioned in previous section 2.4. We will look at two particular examples of specificity measures which are valid only for words (i.e., punctuation is excluded): the entropy of usage and the inverse-document-frequency (IDF) of the word.

More details about these measures are given in the corresponding sections of this chapter. In particular we will describe our own implementation for building the IDF index of words from the Wikipedia online collection of documents.

3.2. Study Case: A Simple Way of Computing Semantic Similarity

Texts are composed of lexical tokens, from which words are most important. They are considered the basic structural subunits for texts. To compare two texts one must first take a look at the words that comprise them. Words can be compared based on their lexical and morphological form, or based on their semantic meaning. Comparing words by their semantic meaning will be discussed in Chapter 4. Then at the next level, one should look at the relations that connect these words in order to properly define the whole meaning of a sentence. The most useful relations between words in a text are the syntactic dependency relations. Comparing these syntactic relations between texts will be discussed in the Chapter 5. For this chapter we limit our discussion to comparing words as single, independent units that form a text, and we only look at their lexical, morphological and part-of-speech forms.

Before working on a research problem it is often recommended for researchers to start with some basic simple solutions and test how well they behave on some evaluation dataset, which is relevant to the research task. This gives the researcher a sense of how difficult or easy a task might be for a given dataset. In our case, a simple approach to calculate the semantic similarity is to assume lexical similarity entails semantic similarity. Of course this is not always the case, but in general and also intuitively, when two texts share most of their lexical tokens, the probability of them being semantically similar is significantly higher. So, in the most simple and naive way of thinking, we regard a text as the unordered set of all the words or lexical tokens that are contained in this text. The problem of comparing two texts is then reduced to comparing the two sets of words or lexical tokens that represent them. A significant part of this chapter will deal with this type of lexical comparing approach. We will present experimental setups and performance results on lexical token overlap methods, enhanced with local and global weighting schemes.

In general, when comparing two sets of elements, there is a similarity metric defined between the elements. The similarity metric can be a very simple one (e.g., a binary defined metric of equivalence, where two elements can be either equivalent or not) or more complex (e.g., based on distance measures, like the edit distance, the linear or the Manhattan distance). A decent baseline for measuring semantic similarity is to choose lexical equivalence as the similarity measure between words. Two words are lexically equivalent if their lexical form is identical; in other words, if they contain the same sets of letters, arranged in the same order.

Given two texts, A and B to compare, a word in A is called a *common* word, if it has an equivalent word in B, and vice-versa. Since the relation of equivalence is symmetrical, it is easy to deduce that the number of common words in A is equal to the number of common words in B. One way to compare two texts, which we represent as unordered sets of words, is to count the number of common words and divide this count by the average number of words from the two sentences, in order to normalize it. We describe this method on the following example:

A. *Mary saw Jimmy leaving the house.*

B. *Mary told Jimmy to go back in the house.*

Based on the representation of meaning we defined in Chapter 2, we represent the two texts as follows[1]:

A. [

(Word=$Mary$, lemma=$mary$, POS=NNP, WN_{SENSE}=1,Deps=(saw:nsubj:-)),

(saw, see, VBD, 1, (-:nsubj:$mary$; -:dobj:$jimmy$; -:obj2:$leaving$)),

($Jimmy$, $jimmy$, NNP, 1, (saw:dobj:-; $leaving$:subj:-)),

($leaving$, $leave$, VBG, 1, (-:subj:$Jimmy$; saw:obj2:-)),

(the, the, DT, 1, ($house$:det:-)),

($house$, $house$, NN, 1, ($leaving$:obj:-; -:det:the)),

(., ., PERIOD, 1, ())]]

B. *[*

($Mary$*,* $mary$*, NNP, 1, (told:nsubj:-)),*

(told, tell, VBD, 4, (-:nsubj:$Mary$*; -:dobj:*$Jimmy$*); -:obj2:go),*

($Jimmy$*,* $jimmy$*, NNP, 1, (told:dobj:-; go:subj:-)),*

(to, to, TO, 1, (go:aux:-)),

(go, go, VB, 1, (-:aux:to; told:obj2:-; -:advmod:back; -:prep_in:house)),

(back, back, RB, 1, (go:advmod:-)),

(in, in, IN, 1, ()),

(the, the, DT, 1, (house:det:-)),

(house, house, NN, 1, (-:det:the; go:prep_in:-)),

(., ., PERIOD, 1, ())]]

For the moment, we choose to not go into the details of how the dependencies are represented above and what do they mean. However, we will make a few

[1] Similarly to the previous example described in Chapter 1, and for esthetic reasons, we chose to omit showing the weighted specificity of the words

comments about this example of a representation. First, note that the lemmas are always in lower case. For words where the notion of a lemma is not applicable (e.g., personal nouns or prepositions) the lemmas are equal to the original words in lower case letters. Second, observe that for the WordNet sense, the only word which has a sense different than 1 is the verb *"to tell"*. As defined in WordNet the fist sense of this verb would be: *"to say something or to express in words"*. But in our case the meaning is more like a command or an order, which is similarly expressed in at the 4th WordNet defined sense: *"to give instructions to or direct somebody to do something with authority"*.

We now continue our discussions of how to apply a simple semantic similarity metric to these texts[2]. Text A has 6 words, out of which 4 are common words that are also contained in Text B. Text B has 9 words, out of which the same number of 4 words are considered common. To calculate the similarity score we divide the number of common words found in both sentences, 4, by the average number of words per sentence, $(6+9)/2 = 7.5$, and we get a final similarity score of 0.533. The score is normalized between 0 and 1, where 0 simply means that there are no common words between the two input texts, and 1 means that the two texts contain the exact same sets of words. Notice that this simple metric does not capture the order of the words in the text. So for example, on the following two sentences, this metric will compute a maximum similarity score of 1, even though their semantic meaning is clearly different.

A. *John shot the sheriff.*

B. *The sheriff shot John.*

[2] For this example we shall ignore the punctuation marks, as opposed to the short example that was previously given in Section 1.5.2

This is one of many cases where the order of words in a sentence and the syntactic relations between them is very important. A simple metric, which ignores these important aspects of a text, will not be able to handle those particular cases. However on the datasets that we experimented on, we noticed that this simple metric, which we have just described, performs fairly good, when compared to other more complex methods that also account for word order and syntax. This suggests that such cases, where syntax and the order of words are critical for measuring semantic similarity, are not that many.

3.3. Similarity Methods based on Lexical Token Overlap

In this section we present various options to expand the simple metric presented in previous section, and acquaint the reader to a range of possible ways to explore the dimensionalities of the input, but only at the lexical level. We shall use only the basic features that are present in our semantic representation, features such as: the original lexical form, the base form (either the lemma or the stem) and the part-of-speech of all words that are in the input texts. When comparing two lists of lexical items, which conform to our representation described in Section 2.4, there are several decisions to make, some of them concern filtering out irrelevant items, while other refer to what information will be used to compare the tokens. The experiments show that, for some decisions, there is no clear winner on what option is best to make. As we will see, every option has advantages and drawbacks. Making the correct choices depends pretty much on the type of task we need to solve and the datasets what we will be working with. In the following paragraphs we try to make a complete list of all these possible choices.

Ignore Punctuation. A very important decision that we need to make when comparing two texts on the lexical level, is what lexical elements that are contained in the text we include in our comparison. For example, do we look at punctuation or do we ignore it? Both choices have advantages and disadvantages. On one hand, one might say punctuation is much less important than words and it can be viewed as noise in the text which can be safely discarded. A punctuation mark does not express any meaning other than marking the changes in tone and intonation that are used in the spoken language. But if we ignore punctuation then we might lose some important changes in the meaning of the sentence. Take for example the following pair of sentences: *I am a business man.* versus *I am a business, man*. We can see that, in this case, the comma makes all the difference between the two sentences who are in fact expressing different facts.

Content Words Only. Another filtering decision which follows the same reasoning as for the punctuation marks is: do we compare content words only? After all, they are the most representative lexical elements for the meaning of texts. Content words are words whose part of speech is either a noun, verb, adjective or adverb. A counter example against using only these types of words for comparation would be the following two sentences which obviously express different meanings: *John is flying to Seattle* versus *John is flying from Seattle.*. For this example the content words are: *John*, *flying* and *Seattle*.

Remove Stop Words. This is a common technique that is used in many Information Retrieval (IR) tasks, when searching for relevant documents to a list of given *key words* (also called *search words*). Stop words are highly frequent words that occur in most of the documents/instances in a collection (i.e., words such as *the* or *in*). In a task of information retrieval, these words are not important and therefore dropped. Our experiments show that, for the problem of semantic similarity assessment, removing the stop words consistently leads to worse results. This suggest that the stop words are actually important for our task and should not be removed during the comparing process. For our experiments we used a list of 423 unique stop words from the Onix Text Retrieval Toolkit[3] (some of the words in the original list were repeated).

Compare base form of words. The next sets of decisions refer to what part of the lexical token do we use to compare to another token. There are two most feasible choices here. We can use either the original, unchanged lexical form of the token, or we can use the base form of words, in which all morphological appendages are stripped from the original lexical form of the word (e.g., a lemma or stem form). Similar to stop words, truncating words to their base form is a common technique used in IR, in order to reduce the vocabulary and thus the dimensionality of the search space. In our case, we have yet to prove if this is beneficial or not for the performance of our text-to-text similarity assessment metrics. An example that would discourage using only the base form of words is the following: *The children are playing in the courtyard.* versus *The child was playing in the courtyard.* Both of these sentences will have the same form after lemmatizing all enclosed words (i.e., *The child be play in the courtyard.*).

Ignore Case. Comparing two lexical tokens can be done either case insensitive, in which we ignore the case of the letters of the tokens, or case sensitive, in which we do not ignore the case of the letters. In general, it is better to ignore the case when comparing lexical tokens. In this way, capitalized common words that are at the start of a sentence, will be matched with their correspondents from the other sentence, which might not be at the start of the sentence, like the word *people* in the following example: *People were having a good time.* versus *Most people were having a good time.* In contrast, ignoring the case of

[3] Retrieved from: http://www.lextek.com/manuals/onix/stopwords1.html

letters can allow for some erroneous matches to occur, as in the case of the words *us* (meaning *we*) and *US* (meaning *United States*) in the following sentences: *They made US prous.* vs. *They made us proud.*

Compare with Part-of-Speech. In English there are many words, or lexical items, that have multiple part-of-speech in which they can be used (e.g., *walk* can be used either as a noun or as a verb, while cold can be used as a noun or an adjective). To increase our confidence of matching the right words with the same part-of-speech, we can add the following extra condition to the matching process: matched words need to have the same part-of-speech. This will help us spot much faster the difference in meaning between the next two sentences: *Trees line the riverbank* versus *The riverbank ends the line of trees*. In this example the lexical token *line* is used both as a verb (in the first sentence) and as a noun (in the second sentence). In the first sentence, it is suggested that *the line of trees* follows the riverbank, or is parallel to the river, while in the second sentence this line is most probably perpendicular or diagonal to the riverbank. The disadvantage for including the part-of-speech in the comparison is that we may miss lexical tokens that convey the same meaning in the two sentences but have different parts-of-speech, due to the difference in sentences' syntax structure. The following example illustrates this particular case: *They had a pleasant walk in the park* versus *They pleasantly walked in the park*. Note that, for this example, we will need to match the base form of the lexical tokens and ignore the parts-of-speech, in order to get the correct matching.

Unigram versus Bigram overlap. After completing all the filtering steps and deciding on which lexical components of a token to use for the comparison, the next decision to be made concerns the way to compare the lexical tokens, that is, using one single token at a time (unigrams) or using two consecutive tokens (bigrams). Bigrams can sometimes prove very useful to look at because they account for part of the syntactic information in a sentence. As Collins (Collins 1996) noted, 70% of the dependencies in English are between adjacent words, meaning that bigrams of consecutive words are able to capture most of the syntactic dependencies in a sentence. One could go even further and look at the number of common tri-grams or quad-grams of lexical tokens, but in this case the space of possible structures to compare becomes too sparse, increasing the risk of not finding any common lexical structures between the input texts.

Measuring the overlap of lexical n-grams to determine the semantic equivalence between texts has been regularly used on problems of automatic *machine translation* (MT) evaluation. Standard MT evaluation metrics have been proposed to determine the performance of MT systems. Using these metrics, the output of the systems is automatically compared against some ideal translations of the input, created by human expert translators. Some of these metrics suggest interesting ways to combine multiple types of n-gram overlap into one measure. For example the BLEU score (Papineni et al. 2001) is a well known MT evaluation metric that is based on computing the geometric mean of n-gram precision. The n-gram precision is calculated as the rapport between the common n-grams found between the MT system's output and the ideal translation, and the number of n-grams present in the system's output. Similarly, the NIST score (Doddington 2002) also uses n-gram precision, but this time, to compute an arithmetic mean, and weights are used to emphasize informative word sequences. Using MT evaluation metrics to determine the semantic similarity at sentence-level has already been proposed by Finch, Hwang, and Sumita (2005) where they investigated various MT evaluation measures, such as BLEU, NIST, WER and

PER. Others researchers got inspired by this work and further expanded the initial approach (Wan et al. 2006; Das and Smith 2009).

3.4. Weighting Schemas for Lexical Token Overlap Methods

So far we discussed about what kind of tokens from the input texts do we choose to compare and what are some methods to compare them. As was suggested in section 3.2, a simple way to compute the similarity score from comparing the lexical tokens is to count how many common tokens exist between the two input texts and then normalize this count by the average number of tokens that exist in both texts. In this section we expand this simple way of computing the scores, by introducing local and global weights, characterizing each of the tokens from the input texts. We compute the weighted overlap score $WSim$ from the list C of common tokens found between two input texts, A and B as shown in Equation 3.1, where $A \uplus B$ means we first consider all the elements in A and then all the elements in B. Notice that in the simple case, when word weighting is not used (either local or global weight), then the corresponding values for these weights are equal to 1. Also, when we compare bigrams, the global weight is computed as the maximum global weight out of the terms that compose the bigram. We also tried to define this as the average global weight between the terms, but found that the performance in this case slightly decreases.

$$WSim(A,B) = \frac{2 * \sum_{w \in C} [weight_{global}(w) * weight_{local}(w)]}{\sum_{w \in A \uplus B} [weight_{global}(w) * weight_{local}(w)]} \tag{3.1}$$

A local weight contains information about the frequency of a lexical token in a text. In our representation, every appearance of a lexical token in the text is accounted for, since we keep an ordered list of all tokens in the text. When we simply count the common lexical tokens found in this list, we are actually using a token *frequency-based* local weighting scheme. Another possibility here, would be to use a *binary-based* local weighting scheme, meaning that we count only once the lexical tokens found in the input texts. For example, on the following pair of sentences: *"To be, or not to be."* versus *"To be alive, or not to be alive."* a frequency-based local weighting scheme will count 6 common tokens (punctuation excluded), while a binary-based local weighting scheme will count only 4 common tokens. The normalization of these counts will also follow the same type of local weighting, resulting a score of 0.86 for the first case, and 0.57 for the second case. Another type of local weighting, which is commonly used when creating LSA spaces (Martin and Berry 2007) is the logarithm of the frequency: $log(frequency + 1)$. The role of this weighting scheme is to decrease the effect of large differences in frequencies, for tokens used in large blocks of texts. For smaller texts however (e.g., sentences), it is much less probable to encounter such tokens with a high local frequency usage. We will use this type of local weighting in the next chapter, when we talk about using LSA vectors to measure the semantic similarity between texts.

Global weighting schemas tell something about how important a word is, when used in a particular sentence, paragraph or document. To measure this factor of importance for words, we use the so-called *specificity* of words. Weighting words based on their specificity has proven to be very useful for Information Retrieval tasks, where the common way to describe this specificity is through the Inverted Document Frequency (IDF) value. Another way to define the specificity of a word is by computing the so-called *entropy* value of the word (more details about these two values will follow). As previously mentioned in Section 2.4, the theoretical assumption in using the specificity of a word for semantic similarity assessment, is that, if a word is considered highly specific (e.g., a less common proper name - *Cleveland*, a rather uncommon noun - *rebarbative*, or a noun that is very specific to a particular topic - *eukaryote*, for the topic of biology), then this word should play an important role when considering for semantic similarity between two texts.

In Information Retrieval related tasks, the IDF is used in combination with the *term frequency* (TF) value of the lexical terms in a document, in order to represent that document as vector in a space, whose dimensions are defined by the vocabulary of terms. The IDF is defined as the logarithm of the normalized inverse of the document frequency: $idf = log(D/df)$, where D is the total number of documents in the collection, and df is the total number of documents where the current term appears. If the term appears in very few documents, it means the term is very specific for those documents and therefore important. In a previous work, Corley and Mihalcea (Corley and Mihalcea 2005) have tried using an IDF index built from the British National Corpus, to solve the problems of entailment and paraphrase identification on the RTE and MSR corpora consequently. For our evaluation, we extracted an index of IDF values from Wikipedia, a very large collection of online documents. We shall describe the process for extracting these values in Section 3.9 at the end of this chapter, along with some relevant statistical data, about the online collection.

The *entropy* of a word is defined as $1 + \sum_j \frac{p_{ij} log_2(p_{ij})}{log_2(n)}$, where $p_{ij} = \frac{tf_{ij}}{gf_i}$, tf_{ij} = type frequency of type *i* in document *j*, and gf_i = the total number of times that type *i* appears in the entire collection of *n* documents (Martin and Berry 2007). This value gives less weight to types that are frequently occurring in the collection of documents, while also taking into account the distribution of the other types over documents (Dumais 1991). In our experiments, we will use the entropy weights calculated from the TASA[4] corpus, same corpus used to derive the LSA space, which we will use in Chapter 4 (where we talk about semantic similarities between words).

3.5. Symmetric versus Asymmetric Semantic Similarity

So far in this chapter, we have discussed simple ways to measure semantic similarity, as a symmetric relation between two texts. Semantic similarity can also be calculated as an asymmetric relation between two texts. *Textual paraphrasing* is an example of symmetric semantic similarity relation, while *textual entailment* is an example of asymmetric semantic similarity relation. By default, when we look at asymmetric relations between two texts, A and B (where A is the first text and B is the second one), we consider only when the asymmetric relations goes from the first text, A, to the second text, B. In case of entailment, we call A - the *Text* and B - the *Hypothesis*.

To measure the asymmetric semantic similarity between texts A and B, we must evaluate if text B is semantically similar to text A (or, for entailment, if text B is entailed from text A). If we choose to do this only through methods of simple lexical overlap, then we must look at the percentage of lexical tokens in B which are also found in A. Therefore, the only difference between computing a symmetric similarity score and an asymmetric similarity score is the way we normalize the count of common tokens found. For a symmetric relation we have already suggested a normalization method, by dividing the count to the average number of total tokens from both input texts. For an asymmetric relation, from

[4] The company that built this corpus was called Touchstone Applied Science Associates

text A to text B, the normalization should be made by dividing the count of common tokens to the number of total tokens that are in text B. This way of measuring the asymmetric similarity is also referred to as computing the *n-gram precision* overlap of lexical tokens (Finch, Hwang, and Sumita 2005). To formally extend this measure to include the weighting schemas, previously described in Section 3.4, we say we compute the weighted asymmetric similarity score $WASim$ from the list C of common tokens found between two texts, A and B as follows:

$$WSim(A,B) = \frac{\sum_{w \in C} [weight_{global}(w) * weight_{local}(w)]}{\sum_{w \in B} [weight_{global}(w) * weight_{local}(w)]} \tag{3.2}$$

3.6. More Options on Lexical Overlap Normalization

Previous sections suggested that, depending on the type of the semantic similarity relations, which is needed for a particular task, different ways of normalization can be applied to the computed weighted lexical overlap score. For example, an asymmetric relation of textual entailment from text A to text B can be assumed by normalizing the weighted overlap score to the weighted lexical score of sentence B. Regarding symmetric similarity relations, we assume in this chapter a normalization by the average weighted score of the two input texts. But there are other ways to normalize the overlap score, in order to obtain measures for symmetric similarity relations. For example, one could normalize on the minimum between the weighted lexical scores of the two inputs, or the maximum (as is shown in Equation 3.3. To make an easier reference of these normalization options in our experiments, we note with *Average-Norm* the normalization on the average length of the input texts, and *Max-Norm* the normalization on the maximum length of the inputs. We will only use these two types of normalization, as we found out that they are consistently more efficient on the task of paraphrase identification.

$$WSim(T_1, T_2) = \frac{\sum_{w \in C} [weight_{global}(w) * weight_{local}(w)]}{Max(\sum_{w \in A | w \in B} [weight_{global}(w) * weight_{local}(w)])} \tag{3.3}$$

3.7. From Quantitative to Qualitative Assessment

We evaluate these simple similarity metrics based on lexical token overlap on the three datasets which were previously introduced in Chapter 1 (the MSR, ULPC and RTE corpora). Since all these datasets assess the semantic similarity using only binary qualitative values (i.e., is semantically similar or not) we need to convert the similarity scores that our metrics generate, into binary qualitative values of the semantic similarity relation that is evaluated by the dataset (paraphrase for MSR and ULPC; entailment for RTE). This can be done easily by deciding on a threshold value that will split the evaluated dataset in two groups: instances that have the calculated similarity score below the threshold value will be considered *negative* instances, meaning they do NOT have the semantic similarity relation defined by the dataset, while instances with similarity values above or equal to the threshold will be considered *positive* instances.

3.7.1 Calculating Similarity Thresholds for Qualitative Assessment

The next question is how to calculate the threshold value. There are a few possible solutions to compute it, by looking at the specifics of the dataset. Some are simple and require minimal information about the dataset, such as the ratio distribution of positive instances, while some are more complex and require a complete supervised learning from a training dataset.

Compute optimum threshold when only data distribution is known. In the simplest case, with a minimum amount of available information about the data, we can compute the threshold by looking only at the distribution of the data (what percentage of instances in the training set are positive). Based on this information, we search for a threshold value that will split the training data in two groups, with a similar distribution of the classified output values (positive vs. negative). The assumption here is that the distribution of instances in the testing dataset is similar enough with the one in the training dataset, since the two sets are initially generated from the same pool of samples.

The procedure for looking up the threshold in the training dataset is as follows. Calculate semantic similarity values for all the training instances. Sort the instances in ascending order, based on the computed values. Then, search for the one instance in this ordered list that splits the list into a distribution equal to the one which characterizes the training data. The computed similarity value of this instance is the threshold we want.

Compute optimum thresholds using supervised learning methods. A more complex method to compute the threshold is through supervised learning. The input is an instance with a single continuous attribute, the similarity value and the output is a binary value. Whether we use perceptrons, threshold classifiers, SVM classifiers or even decision trees, all these methods share one goal in common, to maximize accuracy on the training data. Therefore the output of these classifiers is a threshold value that will give optimum accuracy on the training data.

A simple and straightforward way to calculate this optimum threshold, without using some more complicated learning algorithms is as follows. Similarly to the previous method, first sort the instances in ascending order, based on the similarity metric values. Then gradually select all similarity values from the list, in order, and calculate the accuracy of a system that will use them as thresholds on the training data. Because these values are calculated in ascending order, and because of the nature of the threshold classifier, computing accuracy for all these values can be done on O(n), where n is the size of the training dataset (i.e., the number of training instances). Finally, select the threshold that gives best calculated accuracy on the training dataset.

For our experiments, since we already know much more about the datasets than a simple distribution of the data, we will use the latter type of optimum, threshold-based classification for all the metrics that are being defined in this chapter and the chapters that will follow. We will call this, the *threshold-based classifier*. We will also experiment with another well known classifier, a Support Vector Machine (SVM), that has as input only one feature, our computed semantic similarity measure. Similar to the threshold-based classifier, the SVM classifier searches for one point in this one-dimensional space defined by the input, which will perform just as our threshold, when analyzing new testing data.

3.8. Performance Results on Lexical Token-based Overlap Metrics

In the current section we report results of our evaluation on some of the similarity metrics that were presented in this chapter. We experimented with all our datasets using both symmetric measures, for the MSR and ULPC corpora, and asymmetric measures, for the RTE corpus. We look at two ways to preprocess the data and extract the semantic representations: one using the OpenNLP library and the second one using StanfordNLP. We report performance scores in terms of *accuracy*, *precision* and *recall* on both the training and testing data. *Accuracy* is calculated as the percentage of all instances correctly classified by the system, out of all instances in the dataset (so this includes both positive and negative instances). *Precision* is calculated as the percentage of correctly classified positive instances, out of the total number of positive instances that were classified by the system. And finally, *recall* is calculated as the percentage of correctly classified positive instances, out of the total number of positive instances that are present in the dataset. All these values are reported as percentage values with three rounded decimals after the point (e.g., $.671$ and $.548$ will approximate to 67% and 55%, correspondingly). We will also indicate the optimum threshold values that were found on the training set and then evaluated on the test set.

For aesthetic reasons we will use letter codes to describe all the options that are used in each of the methods listed. The first letter describes the filtering of tokens: P means we compare all tokens, including punctuation; W means we exclude punctuation and compare only words; C means we compare content words only; S means we use all words, excluding the stop words. Second letter tells how we compare the tokens: W means we compare them by their original raw form, B means we compare them by their base form, while P means we compare only words that have the same part-of-speech and same base form[5]. Third letter, tells whether we used case sensitive (S) or insensitive (I) for the comparison. Fourth letter tells if we used unigrams (U) or bigrams (B) of consecutive tokens to compare. Fifth letter refers to the global weight used, whether is IDF-based (I), entropy-based (E) or none at all (N). Sixth letter indicate the local weight used and there are two options here: word type frequency (F) or

[5] We also looked at comparing words with same part of speech and same word form, but the results were weaker and very close; the close proximity of the results can be explained by the fact that the morphological information that is being stripped in the base form, is actually encoded in the POS

no local weighting (N). Therefore the name of any method that we experiment with in this chapter conforms to the following notation:

$$MethodName = (P|W|C|S).(W|B|P).(S|I).(U|B).(I|E|N).(F|N)$$

From this notation we see that there are a total of 288 possible combinations (4 x 3 x 2 x 2 x 3 x 2) that we can experiment with, on one dataset and one type of preprocessing, resulting in a total of 1728 possibilities to explore on all 3 datasets (MSR, ULPC, RTE) and the 2 types of preprocessing (OpenNLP and StanfordNLP). It is obviously a rather assiduous task to experiment with all these possibilities, so we need to do an intelligent search, gradually analyze each option and see which one works better for a specific corpus, then use it in combination with the other options.

We start by looking at the MSR corpus, preprocessed with OpenNLP-based functions. We report performance scores on some of the lexical token-based overlap methods in Table 3.1. We then discuss on these results and explain why some of the variants were omitted.

For this initial set of experiments, we found out that a simple combination of `P.B.I.U.N.N.` (include all tokens, compare unigrams of base forms, ignore case, and do not use any weighting) works best for the testing part ($.7403$ accuracy) although on the training part the results are actually lower ($.7299$ accuracy). We also found out that a combination of *words only* with *compare word form* also shows promising results (`W.W.I.U.N.N` gives $.7357$ accuracy on test data). Comparing tokens using case sensitive mode slightly decreases the accuracy scores on the test data (although it gives best precision in one case - `P.B.C.U.N.N`), so does using same part-of-speech restriction to compare between tokens. We can also see that if we compare only content words, or only words that are not stop-words then we

Table 3.1
Lexical methods on MSR, with OpenNLP parsing and Average-Norm

		Performance on Train			Performance on Test		
Method	Threshold	Acc.	Prec.	Recall	Acc.	Prec.	Recall
P.W.C.U.N.N.	.5517	.7223	.7529	.8765	.7264	.7517	.8788
P.W.I.U.N.N.	.5500	.7230	.7408	.9074	.7333	.7438	.9137
P.B.C.U.N.N.	.5957	.7282	.7730	.8460	.7258	**.7705**	.8370
P.B.I.U.N.N.	.5660	.7299	.7494	.9016	**.7403**	.7538	.9050
P.P.I.U.N.N.	.5417	.7154	.7488	.8707	.7125	.7457	.8614
W.P.I.U.N.N.	.5106	.7176	.7475	.8787	.7165	.7452	.8718
W.B.I.U.N.N.	.5490	.7311	.7624	.8743	.7287	.7539	.8788
W.W.I.U.N.N.	.5161	.7252	.7437	.9052	.7357	.7470	.9111
W.W.C.U.N.N.	.5106	.7230	.7500	.8849	.7351	.7563	.8875
C.W.I.U.N.N.	.5161	.6943	.7373	.8503	.6916	.7331	.8431
C.B.I.U.N.N.	.5385	.7061	.7467	.8547	.6962	.7354	.8483
S.W.I.U.N.N.	.4375	.6953	.7078	.9346	.6997	.7049	.9433
S.B.I.U.N.N.	.5333	.7004	.7446	.8471	.7113	.7479	.8535
P.B.I.B.N.N.	.2642	.6896	.7155	.8972	.6968	.7161	.9015
P.B.I.B.N.F.	.2642	.6894	.7154	.8968	.6974	.7166	.9015
P.B.I.B.E.N.	.2119	.6828	.7020	.9215	.6887	.7015	.9259
P.B.I.B.I.N.	.1872	.6872	.7037	.9274	.6783	.6947	.9207
W.W.I.B.N.N.	.1739	.6882	.6960	.9557	.6904	.6941	**.9555**
P.B.I.U.N.F.	.5778	.7289	.7626	.8692	.7362	.7629	.8753
W.W.I.U.N.F.	.5000	.7242	.7355	.9241	.7351	.7399	.9276
P.B.I.U.E.N.	.5422	.7061	.7218	.9190	.7107	.7201	.9241
P.B.I.U.I.N.	.5274	.7058	.7209	.9212	.7003	.7126	.9207
W.W.I.U.E.N.	.5487	.7036	.7246	.9052	.7038	.7190	.9102
W.W.I.U.I.N.	.5833	.7034	.7403	.8638	.6986	.7352	.8544

Table 3.2
Lexical methods on MSR, with Stanford parsing and Average-Norm

		Performance on Train			Performance on Test		
Method	Threshold	Acc.	Prec.	Recall	Acc.	Prec.	Recall
P.B.I.U.N.N.	.5806	.7316	.7548	.8925	.7368	.7572	.8893
W.W.I.U.N.N.	.5098	.7272	.7398	.9194	.7299	.7383	.9198
S.B.I.U.N.N.	.5333	.7085	.7432	.8685	.7049	.7370	.8649
P.B.I.U.N.F.	.5778	.7321	.7609	.8798	.7310	.7566	.8779
P.B.I.U.E.N.	.5761	.7112	.7349	.8954	.7130	.7349	.8893
P.B.I.U.I.N.	.5439	.7100	.7245	.9208	.6968	.7143	.9067
P.B.I.B.N.F.	.2609	.6931	.7148	.9077	.6916	.7099	.9067

have lower performance, although removing only stop-words seems to have less detrimental effect on performance. This suggests that function words (e.g., articles, conjunctions, auxiliary verbs) are actually important for the problem of semantic similarity assessment. This was also agreed by Li and colleagues (Li et al. 2006), that function words, such as *the*, *of*, *an*, are actually important for sentences because they carry structural information. We also notice, from our results, that comparing texts by using only bigrams seems to have a detrimental effect on the accuracy and precision, but it managed to give best recall in one case (`W.W.I.B.N.N`). We also see that the optimum threshold found is much lower than when using unigrams. This is explained by the fact that we are finding much fewer commonalities between the input texts, when comparing at bigram level. As an extension to these methods, one might try to combine the unigram and bigram overlap to see if the scores improve, similarly with some of the MT evaluation metrics (e.g., BLUE). We will look at this type of metric later in the dissertation. Finally, concerning the weighting schemas, we found out that in general performance is decreased when using any kind of weighting, less on the local weighting and much more on the global weighting. When we compare the global weighting schemas, we see that using entropy-based weighting gives better scores than IDF-based weighting.

For our next experiment, we selected a few good and representative methods, based on the accuracy reported on the test part in Table 3.1, and then we tested them on MSR again, but this time preprocessed with StanfordNLP. The results for these methods are listed in Table 3.2. Noticed that the results on data preprocessed with the OpenNLP are consistently higher on the testing part, than the data preprocessed with StanfordNLP, although from the results reported on

Table 3.3
Lexical methods on MSR, with Stanford parsing and Max-Norm

		Performance on Train			Performance on Test		
Method	Threshold	Acc.	Prec.	Recall	Acc.	Prec.	Recall
W.W.I.U.N.F.	.4828	.7294	.7589	.8783	.7409	.7624	.8867
W.B.I.U.N.F.	.5263	.7316	.7783	.8427	.7310	**.7756**	.8378
W.W.C.U.N.F.	.4615	.7274	.7497	.8954	.7310	.7491	.8954
W.B.C.U.N.F.	.5000	.7289	.7546	.8870	**.7432**	.7600	.8971
P.W.I.U.N.F.	.5238	.7291	.7632	.8685	.7333	.7669	.8605
P.B.I.U.N.F.	.5238	.7282	.7533	.8885	.7397	.7616	.8858
P.B.C.U.N.F.	.5238	.7269	.7567	.8780	.7386	.7656	.8745
W.B.I.B.N.F.	.1818	.6911	.7001	.9495	.6974	.7007	.9512

training, we see that with StanfordNLP we can do better learning, at least on the training part.

We do not want to leave a misguided impression to readers that OpenNLP preprocessing is better than Stanford preprocessing. As a matter of fact, in our next chapter, where we deal with word-based semantics, we shall see that using a Stanford-based preprocessing, clearly outperforms the OpenNLP parsing. In consequence we decided to do more testing with the Stanford parser, using different normalization factors and other various preprocessing options. In Table 3.3 we report 8 cases of lexical overlap-based methods, when using Stanford preprocessing and a normalization factor based on the maximum length of the two input sentences, as in Equation 3.3. What we found out here, is that using frequency of tokens for the local weighting scheme, consistently give better accuracy results on the test part. Moreover, we found case where Stanford performs actually better than OpenNLP, regarding accuracy (.7432 on method W.B.C.U.N.F. and precision (.7756 on method W.B.I.U.N.F.).

Table 3.4
Lexical methods on ULPC, with Average-Norm

		Performance on Train			Performance on Test		
Method	Threshold	Acc.	Prec.	Recall	Acc.	Prec.	Recall
OpenNLP Processing							
P.B.I.U.N.N.	.4211	.6411	.6139	.8918	.6253	.6091	.8791
W.W.I.U.N.N.	.3571	.6424	.6173	.8769	.6353	**.6194**	.8645
S.B.I.U.N.N.	.4286	.6484	.6193	.8943	.6293	.6106	.8901
P.B.I.U.N.F.	.4118	.6438	.6174	.8831	.6092	.6016	.8462
P.B.I.U.E.N.	.4884	.6518	.6199	.9067	.6092	.5995	.8608
P.B.I.U.I.N.	.5194	.6484	.6201	.8893	.6132	.6026	.8608
P.B.I.B.N.F.	.1143	.6237	.6083	.8383	.6072	.6066	.8022
StanfordNLP Processing							
P.B.I.U.N.N.	.4615	.6471	.6224	.8694	.6192	.6113	.8352
W.W.I.U.N.N.	.3846	.6431	.6213	.8570	.6253	.6168	.8315
S.B.I.U.N.N.	.4000	.6498	.6144	.9316	**.6433**	.6156	**.9267**
P.B.I.U.N.F.	.4082	.6444	.6163	.8930	.6212	.6082	.8645
P.B.I.U.E.N.	.5414	.6531	.6282	.8657	.6112	.6070	.8205
P.B.I.U.I.N.	.5302	.6524	.6202	.9080	.6333	.6148	.8828
P.B.I.B.N.F.	.1143	.6231	.6072	.8420	.6112	.6082	.8132

We next evaluated the initial set of methods that were selected to evaluate the MSR corpus with StanfordNLP preprocessing (while using average sentence length normalization) on the two other corpora, using both types of preprocessing and average length normalization. Table 3.4 shows performance scores on the ULPC corpus, while Table 3.5 shows performance results on the RTE corpus.

We observe that the ULPC corpus tells a different story than MSR. Best accuracy and recall on test data is achieved with StanfordNLP preprocessing, where stop words are removed, while best precision is achieved on OpenNLP preprocessing with all words included. We also note an improvement on StanfordNLP preprocessing, when using global IDF weighting, in comparison to using entropy global weighting or no weighting at all. For the RTE corpus, we see

Table 3.5
Lexical methods on RTE, with asymmetric normalization ($A \rightarrow B$)

		Performance on Train			Performance on Test		
Method	Threshold	Acc.	Prec.	Recall	Acc.	Prec.	Recall
OpenNLP Processing							
P.B.I.U.N.N.	.5909	.6039	.5718	.8470	.6100	.5817	.8512
W.W.I.U.N.N.	.5556	.5986	.5749	.7757	.6150	.5985	.7561
S.B.I.U.N.N.	.5833	.6095	.5789	.8206	.6475	.6131	.8463
P.B.I.U.N.F.	.6000	.6045	.5727	.8422	.6112	.5826	.8512
P.B.I.U.E.N.	.6850	.6119	.5839	.7953	**.6550**	.6184	**.8537**
P.B.I.U.I.N.	.7385	.6063	.5876	.7293	.6338	.6085	.8000
P.B.I.B.N.F.	.2000	.5888	.5698	.7451	.6025	.5931	.7146
StanfordNLP Processing							
P.B.I.U.N.N.	.5909	.6042	.5712	.8549	.6150	.5850	.8561
W.W.I.U.N.N.	.6154	.6005	.5899	.6755	.6275	**.6267**	.6756
S.B.I.U.N.N.	.6250	.6114	.5846	.7858	.6400	.6147	.7976
P.B.I.U.N.F.	.6190	.6074	.5780	.8132	.6187	.5933	.8146
P.B.I.U.E.N.	.6930	.6092	.5811	.7995	.6463	.6132	.8390
P.B.I.U.I.N.	.7386	.6076	.5851	.7562	.6288	.6033	.8049
P.B.I.B.N.F.	.2000	.5893	.5702	.7456	.5975	.5891	.7098

that using entropy global weighting on lemma and punctuation unigrams gives us best accuracy and recall with OpenNLP preprocessing, while best precision is gained from a StanfordNLP preprocessing, using all words and no weighting.

We remind the reader, that the normalization for RTE differs that the other two corpora, in the sense that we are looking for an asynchronous relation of semantic similarity from A (first sentence) to B (second sentence). To show that this normalization actually makes sense for this corpus, we present in Table 3.6 non-standard experiments where we used on the RTE corpus, with OpenNLP processing, the other two possible types of normalization, one for the symmetric relation of semantic similarity (used on MSR and ULPC) and the other for the inversed asymmetric relation from text B to text A). It can be observed that the

Table 3.6
Lexical methods on RTE, with OpenNLP parsing and non-standard normalization

		Performance on Train			Performance on Test		
Method	Threshold	Acc.	Prec.	Recall	Acc.	Prec.	Recall
Symmetric Relation ($A \leftrightarrow B$) with Average-Norm							
P.B.I.U.N.N.	.4727	.5439	.5865	.3166	.5400	.6193	.2659
W.W.I.U.N.N.	.3051	.5484	.5486	.5773	.5987	.6282	.5317
S.B.I.U.N.N.	.3158	.5630	.5535	.6797	.6313	.6275	.6902
P.B.I.U.N.F.	.3333	.5394	.5409	.5588	.6050	.6313	.5512
P.B.I.U.E.N.	.3355	.5466	.5342	.7699	.5938	.5822	.7341
P.B.I.U.I.N.	.4462	.5426	.5493	.5061	.5863	.6165	.5098
P.B.I.B.N.F.	.0444	.5604	.5403	.8454	.5850	.5672	.8024
Inverse Asymmetric Relation ($A \leftarrow B$)							
P.B.I.U.N.N.	.3529	.5251	.5526	.2939	.5300	.6104	.2293
W.W.I.U.N.N.	.1277	.5328	.5225	.8285	.5888	.5725	.7805
S.B.I.U.N.N.	.1667	.5442	.5316	.7905	.6162	.5925	.8049
P.B.I.U.N.F.	.3448	.5214	.5520	.2575	.5225	.6061	.1951
P.B.I.U.E.N.	.2015	.5275	.5194	.8106	.5713	.5595	.7683
P.B.I.U.I.N.	.1782	.5264	.5174	.8718	.5725	.5552	.8341
P.B.I.B.N.F.	.0364	.5551	.5401	.7784	.5975	.5846	.7415

performance numbers reported in this last table are clearly much lower than the standard normalization which we used in Table 3.5.

Regarding the performance scores on training versus testing we see that, on the MSR and RTE corpora, most of the methods, which offer good results on the testing data, have actually lower performance scores on the training data. This clearly indicates that these methods, due to their simplistic nature, do not tend to overtrain and that they can generalize pretty well on unknown data.

As a general conclusion on the experiments presented so far in this chapter we can conclude that choosing the best combination of lexical token overlap is not always an easy task and it depends much on the type of the data that is being worked on. The only variants that constantly gave poor performance results were when comparing bigrams of lexical tokens, or when looking at only content words. This is somewhat expected, because when we rely only on bigrams to detect similarities between texts we are ignoring a big chunk of useful information that could be hidden in unigram-to-unigram relations. This inconvenience can be partially solved by using unigrams and bigrams in combination to detect semantic similarities between texts.

Relating to other previous work that has been done on these tasks of paraphrase and entailment recognition, we note that these simple methods, based on lexical token overlap are working quite well. For example, looking at the accuracy, on the MSR corpus, our best method (`W.B.C.U.N.F.` with .7432 accuracy in Table 3.3) is only less than 2% lower than the previously reported state of the art methods (Androutsopoulos and Malakasiotis 2010); while on the RTE corpus our best method (`P.B.I.U.E.N.` with .6550 accuracy) actually outperforms the best system reported in (Heilman and Smith 2010) with more than 1%, although the precision is rather quite low in our case (between 3% to 9% lower).

3.9. Computing IDF values from Wikipedia

In this chapter we have proposed various ways to use the semantic representation presented in previous chapter, to compare and compute the lexical overlap between two given texts. In Section 3.4 we presented ways to compute the similarity measure between the inputs, by using different local and global weighting schemas for the lexical tokens. One of the global weighting schemas that we suggested is based on a computed IDF index of lexical terms. In this section we present our own method to extract the IDF index from a very large collection of documents. Although we use the index to enhance and improve our methods of measuring the semantic similarity between texts, this index has the potential to be useful in other types of NLP and IR related tasks.

For our experiments, we use Wikipedia, the largest and most diverse collection of documents freely available on the Internet, as the source for calculating the IDF values. We built this index initially for the task of semantic similarity assessment, as inspired by the work of Corley and Mihalcea (2005), who used an IDF index built from the British National Corpus to use on the task of paraphrase identification. The IDF values are calculated from the DF (document frequency) values extracted from over 2.2 million Wikipedia documents. To account for the data sparseness factor, augmented by the very high number of documents available, and keep the normalization of the IDF values reasonably simple, we will assume that a word can only appear in a maximum of 1 million (10^6) documents collection. All DF values that exceed the maximum number of documents are reduced to the maximum accepted value of 10^6. This means that for those several words that actually appear in more than 1 million documents in the Wikipedia collection we will assume the same minimal IDF value of 0. This also means that the maximum IDF value, on words that appear in only one document, is $\log(10^6) = 6$. Therefore, to normalize the values of our computed IDF index, we would divide them by 6, the maximum possible, non-normalized IDF value.

For the rest of this section we present details on the process of computing the IDF values. Besides IDF, we also studied Zipf and Heaps laws. Zipf law is well explained in (Manning and Schutze 1999). For more details on the Heaps law and usage of IDF values in Information Retrieval we refer to the book of (Baeza-Yates and Ribeiro-Neto 1999). Our analysis confirms that Wikipedia collection manifest the same statistical data as other text collection previously studied by researches.

In addition, we approximated the parameters for these functions, which we consider to be specific to the Wikipedias word collection (as it was at the beginning of 2008).

3.9.1 Data Preprocessing

As for the task of semantic similarity assessment, data preprocessing is very important to the process of building the IDF index. At specific points in time, all current pages from Wikipedia are being saved (or dumped) to huge XML files which we call Wikipedia database dump files. Each page is stored in an XML node, called *page*. The title of the page is stored in the *title* sub-node, and the content of the page is saved in the *text* sub-node, which is located inside the *revision* sub-node of the *page* node. For each page we extract the title and textual content from the database dump file.

The main issue we had with processing the content of these pages is that this content is full of extra information used to display the pages in a more presentable form to the readers. This type of info is not really useful for our task where we are only interested in the textual content that is presented to the readers. The extra information is delimited from the rest of the text by using two main types of enclosing tags: the wiki tags and the html tags. All content enclosed by these tags had to be removed, before applying further operations to the textual content encoded in the Wikipedia pages. Below are detailed some of the critical steps that we did to preprocess the documents:

1. Remove any non printable characters, except the carriage-return character (”\n”) and convert back to normal form some of the XML special characters ($>$, $<$, ” , &) that are required in future steps to remove content encoded in XML tags. In some cases, the ”greater than” and ”less than”

characters were double encoded, so after replacing back the *ampersand* character, we had to try and restore these characters again.

2. Remove any HTML comments that are not displayed to the normal reader[6]

3. Remove any wiki tables that are stored in the wiki pages. The information stored in these tables contains mostly numbers and enumerations that we considered as not being part of the real textual content. If some of the data represented in the tables is indeed important, then it is usually also mentioned in the textual part of the page. Therefore any content that is included within the special wiki table tags delimiters is removed from further processing[7]. For similar reasons, also remove any information contained in the HTML *table* and *gallery* tags.

4. Remove any information enclosed in the double braces special wiki tags. This is usually represented by different Wikipedia variables (e.g., CURRENTMONTH) or other types of information that has special meanings and is not displayed as a normal text to the wiki reader.

[6] For some of the pages, the standard HTML comment rules are not followed: we encountered cases where we had opening comment tags but the closing tags were missing; as a consequence the rest of the wiki page should have normally been considered as html comment. But as we noticed, the wiki parser has some different rules when parsing enclosed html data. When a new section begins in the wiki page (which is marked by a line that begins with at least two equal signs - ==), any previously unclosed HTML comments are ignored, and the following text or section is seen as a normal, uncommented text. We have applied the same parsing rule when extracting the HTML comments. This rule is also valid for other wiki enclosing tags that will be described in the following steps.

[7] Again, for special cases, as mentioned in the previous note, we apply the same special treatment when an enclosing wiki table tag is missing

5. We give special treatment to strings of the form: *[[label:text]]* where *label* describes the type of information that is stored in the *text* part. Depending on the type of the *label*, the wiki parser displays different outputs of the *text*. The most important type of *label* is probably the *image* label, which displays an image in the current page; and below the image, a textual description for this image. In such cases, we keep the image description and remove the rest of the information contained in the string. Other types of labels include *categories* or *cross-language*, for special links; these labels will not be shown to the wiki reader and therefore we decided to eliminate them from our preprocessed text. The last type of label that we handle is when the *text* string contains the *pipe* character. In such cases, we found out that the text after the *pipe* is usually being displayed in the web page. For example, this is the case when a word in the page is linked to another word in the wiki dictionary. Therefore we will keep the text that follows the *pipe* character.

6. Remove any http links and references that are included in special wiki enclosing brackets, but keep the description of the links in the text, whenever they are available.

7. At this step we remove more HTML tags that do not contain relevant textual content or do not appear on the page: *math, sup, tt, div, span, font, br, ol, http, blockquote*. Also restore any HTML encodings of non-breaking space characters (&nbsp) to the space character.

8. Remove from the text all wiki links enclosures, which are marked by two double square brackets (as in ”*[[wiki-link]]*”; the *wiki-link* is not removed; only the brackets). The reason for doing this step is that there are words in

Wikipedia which have only a partial part of their lexical form marked as a wiki link. Therefore, by removing the brackets we restore the full lexical form of those words.

9. Replace all the special characters except the *dash* (-), with a space character. The *dash* might be contained in some complex words and, as we will show in our results, some of these words are important to be kept in their full complex form. We replace the *dash* character only if it is outside the words or if there are multiple successive instances of it in the text.

10. As a final step, we convert all letters to lower-case.

From executing all steps enumerated above, we are left with a massive list of documents that contain only relevant textual content, with words in lower-case and numbers separated by one or more spaces, and the dash character. We further ignore the numbers (strings of digits without any letter in their lexical form) and leave any digits that are connected to tokens that also have letters in their lexical form. As we found out, there are certain complex words that contain digits and are frequently used in the literature (e.g., *3-dimensional*, *7-eleven*, *16-bit*).

The last factor which we consider during the preprocessing phase is the elimination of some pages that are related to Wikipedia's internal organization. These pages include redirects, image description pages, templates, category pages, and pages used for the internal administration of Wikipedia. We think that by including these pages we risk of introducing a bias on type of textual content that is present in Wikipedia. Wikipedia has been commended to be one of the most informative, universal and unbiased collection of document available online so far, and so we would want to keep this trait still truthful.

Table 3.7
Top document frequency values for words that begin with letter/digit

a-i	Doc.freq.	k-p	Doc.freq.	r-w	Doc.freq
in	1,989,968	of	1,987,234	the	2,076,714
a	1,972,894	on	1,270,135	to	1,617,516
and	1,827,040	links	975,865	was	1,328,853
is	1,771,390	one	709,560	s	1,184,194
for	1,334,793	or	666,671	with	1,180,794
by	1,273,100	new	594,233	that	902,551

3.9.2 Statistical results

We calculated the IDF values for all the lexical tokens in the Wikipedia collection, as it was at the beginning of 2008. We considered a lexical token to be a string of characters without any spaces, containing at least one letter and might also contain digits and the *dash* (”-”) sign. The only changes to the original texts were described in the previous subsection (we have not applied any stemming or other similar transformation functions to the words).

After the preprocessing phase, we obtained a collection of *2,225,726* documents, from which a total of *5,237,779* are considered distinct words, with the restrictions mentioned before. Table 3.7 lists the most common (or less specific) words used in Wikipedia, and their associated *document frequency* values.

In Figure 3.1, we report a graph on the distribution of words in Wikipedia, based on their first letter. If a word begins with a digit we include that in the category noted with ”_”. Because the numbers of words beginning with the letters *j,q,x,y* and *z* were much lower than the rest, we combined their sets together and report the total count number under the *jqxyz* category.

We looked at a couple of reasons why the number of distinct words is so high. We suspect that this is mainly because Wikipedia contains a lot of proper nouns,

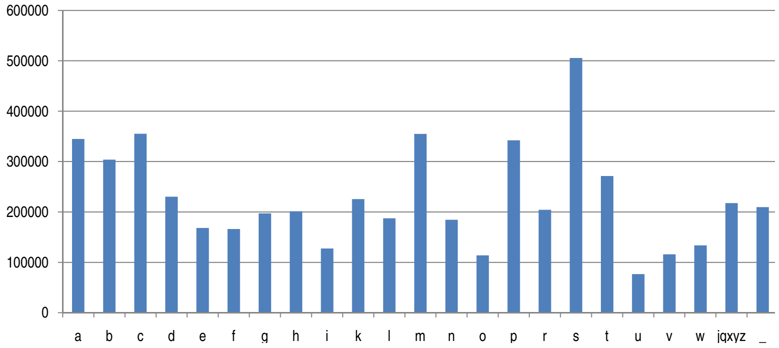


Figure 3.1
Number of distinct words in Wikipedia grouped on the first character

which were also included in our counting. Since Wikipedia is a universal collection of informative documents, we expect that every name used by mankind that is somewhat worth mentioning in an encyclopedia, will be part of this collection. Also, words were not transformed to their base form, and there are many complex words (e.g., two or more words separated by dash characters) that were counted as separate words. We argue that complex words are equally important when building an IDF index and to support this, we list here some of the most common complex words we found on Wikipedia, along with their associated *document frequency* values: *non-families* (36,588), *well-known* (31,296), *so-called* (18,866), *non-profit* (13,062), *short-lived*(12,411) and *full-time* (12,101). Also, sometimes words that were misspelled are counted more than once.

One way to reduce the vocabulary of the LSA index by a significant amount, and not affecting the LSA values too much is to add a restrictive condition to include only words that appear in at least two documents in the Wikipedia collection. This restriction greatly reduces the number of distinct words by a factor of *2.47*. By applying this condition we obtain a total of *2,118,550* distinct words, with the distribution of words based on their first letter very similar to the

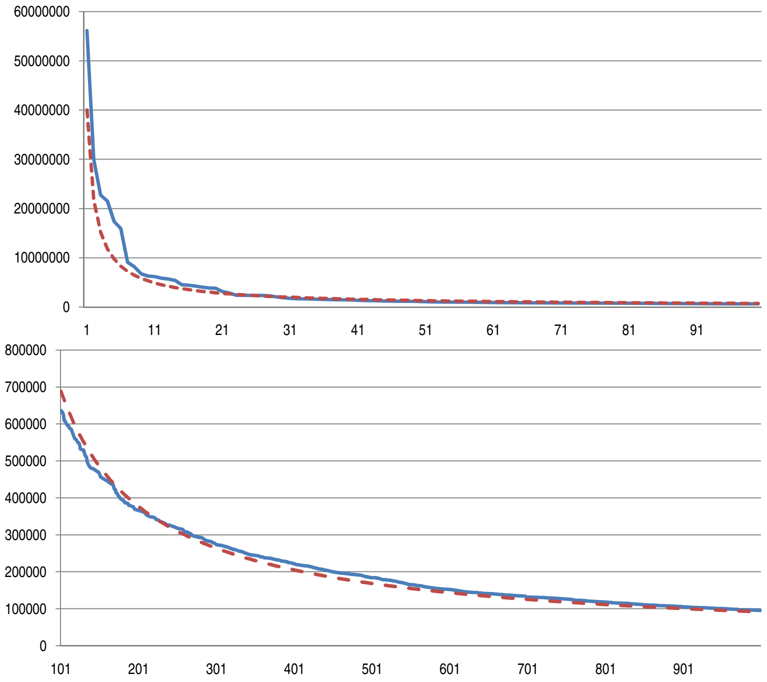


Figure 3.2
Zipfian distribution on the most frequent 1000 words in Wikipedia collection

one reported in Table 3.1, with one exception on the set of words that begin with a digit, where the number is reduced by a factor of *3.58*.

In Figure 3.2 we report the Zipfian distribution for the top 1000 most frequent words found in our preprocessed Wikipedia collection. The Zipfian values were calculated by counting the frequency of every word that was used in collection. This also includes how many times a word appears in one document. The top 5 most frequent words in the collection are: *the, of, and, in, a*. To have a better view of the distribution, we display it in two graphs, one for the first 100 words, and the second for the next 900 words. We also empirically approximate the values of the two parameters *a* and *b* from the Zipfs Law function (Zipf(n) = a / n^b). The solid line represents the Wikipedia distribution for top 1000 words and the dotted line represents the Zipf's Law function for *a = 40,000,000* and *b = 0.88*.

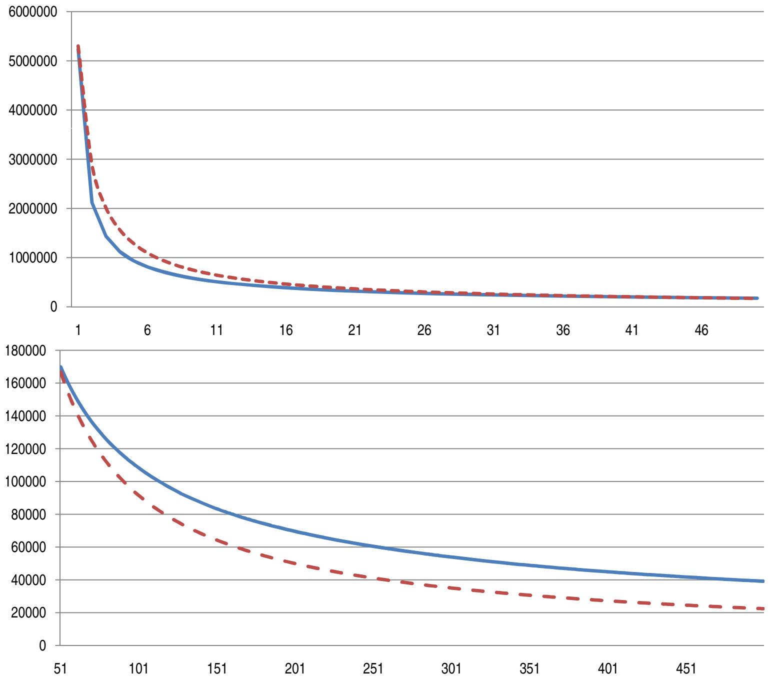


Figure 3.3
Zipfian distribution on the H(n) function for first 500 points

To remain in the Zipfian distribution context, in Figure 3.3 we show an interesting distribution, calculated from the *document frequency* of words, that looks similar to our previously calculated Zip's Law. We start by counting all words that appear in at least one document in the collection, then counting all words that appear in at least two documents, then, words that appear in at least three documents, and so on. We define the function H(n) that will tell how many words appear in at least *n* documents in the Wikipedia collection. Figure 3.3 report the distribution of this function with a solid line, for the first 500 values of *n*. Like before, we approximate the function with a Zipfian distribution, represented with a dotted line, for the following parameter values: *a = 5,300,000* and *b = 0.88* (interestingly note that *b* is same as before).

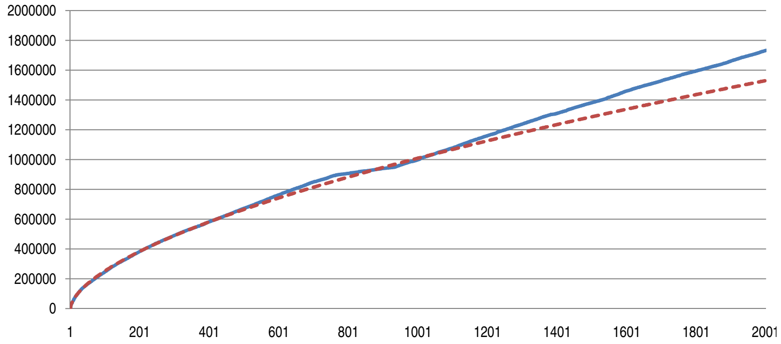


Figure 3.4
Heap's Law distribution on the Wikipedia collection (for the first 200 million words)

Lastly, we report the Heap's Law distribution for the Wikipedia collection. This function tell us how the vocabulary increases in size, proportional to the size of the collection. When we calculate the size of the collection, we take all documents in the collection, combine them together in one single document and count the number of words used so far in the text. The values for the Heap's Law distribution are calculated using the formula: $V_{a,b}(n) = a * n^b$, where n is the size of the text, and a and b are free parameters, which we determine empirically. Figure 3.4 shows the Heap distribution (with a solid line) for the first 200 million points of the function (incremented by a factor of 10,000). With a dotted line we show the function calculated with Heap's Law where the free parameters have the following values: $a = 16$ and $b = 0.6$.

3.9.3 Implementation

We created Perl scripts, to parse and preprocess the Wikipedia database dump file. From this step (which took about 3 hours on a 3.40GHz Pentium system) we managed to reduce the initial size of the file (of approx 14GB) to half. Extracting the IDF index (i.e., counting the *document frequency* for all words) took 2 full days.

CHAPTER 4

WORD-TO-WORD SEMANTICS

4.1. Introduction

The idea of lexical token matching, presented in previous chapter, can be expanded by using some word-to-word (W2W) semantic similarity metrics instead of simple word matching or synonymy information in a thesaurus as in (Qiu, Kan, and Chua 2006; Rus et al. 2008a). The word-to-word similarity metrics can identify semantically related words even if the words are not identical or synonyms. A good package to calculate these metrics is the WordNet similarity package (Pedersen, Patwardhan, and Michelizzi 2004). WordNet similarity metrics rely on statistical information derived from corpora and lexico-semantic information from WordNet (Miller 1995), a lexical database of English. The basic idea behind the WordNet similarity metrics is that the closer the distance in WordNet between words/concepts is, the more similar they are. For instance, in the following example taken from MSR corpus (Dolan, Quirk, and Brockett 2004) semantic relationship between the words *insist* and *say* cannot be established using simple direct matching or synonymy. On the other hand, there is a relatively short path of three nodes in WordNet from *say* to *insist* via *assert*, indicating *say* and *insist* are semantically close.

> Text A: *York had no problem with MTAs insisting the decision to shift funds had been within its legal rights.*
> Text B: *York had no problem with MTAs saying the decision to shift funds was within its powers.*

Another measure to approximate word-to-word (W2W) semantic similarity, which has been extensively used in the literature (Dumais 1991; Nakov, Popova, and Mateev 2001; Graesser et al. 2007; McNamara, Boonthum, and Millis 2007; Dessus 2009) is Latent Semantic Analysis (in short, LSA). In Chapter 2, Section 2.4.2, we introduced LSA as a component of our semantic representation structure. We presented useful properties and drawbacks for using this type of vectorial representation to represent the meaning of words. We also detailed on how to extract the LSA vectors and construct an LSA space from a given collection of documents. In this chapter we present two different methods of how to use LSA when computing semantic similarity between texts. In the first method we use LSA the same way as when using WordNet Similarity metrics (further detailed in Section 4.5), to match and pair semantically similar words. We use the cosine between two LSA vectors to compute the semantic similarity between their corresponding words. The second approach (see Section 4.8) makes use of the LSA vectorial representations for words in a text. We combine these vectors through a weighted sum to create a representative vector for the whole text. Then, likewise as for the word level, we compute the semantic similarity between the two texts using the cosine between their representative vectors. In particular, we investigate the impact of several local and global weighting schemas on Latent Semantic Analysis' (LSA) ability to capture semantic similarity between two texts.

When comparing to our previous methods of simple lexical overlap, we see that by including word-based semantics to measure semantic similarity between texts, we are faced with new challenges. The main challenge here would be to find a cheap and effective way to pair words. Suppose we have the following two input texts that, in most contexts, should convey the same message:

Text A: *My pet enjoys playing with your dog.*

Text B: *My cat likes to play with your pet.*

For our previous methods of simple lexical matching, the problem of pairing is trivial: find any two identical words from both texts and pair them; then find another two words, and so on. Using this method in our example we find that the following words can be paired: *my, pet, with, your* and *play*, if we use the base form of words. Ideally, we should also be able to pair *enjoys* with *likes*, which can be easily done by using synonymy information. The real problem we are faced with in this example is regarding the words *dog* and *cat*, which should be paired with their correspondent *pet*s from the other sentence. However, if we look at pairing identical words first, as in a greedy approach, then the two *pet* words (from both sentences) will be already paired, by the time we are trying to find pairs for *cat* and *dog*. To solve this issue, we should look at all words in the text and try to find the optimal matching, according to a particular function of word-based semantic similarity. Word-based semantic similarity metrics provide normalized values that can tell, on a scale of 0 to 1, how close two words are in their meaning. Identical words and synonyms are considered to have a word-to-word similarity of 1. For our example, we would need a metric that has preference on pairing *cat-pet* and *dog-pet*, instead of *pet-pet* and *cat-dog*. Unfortunately, most metrics that will be studied in this chapter, are not able to find such subtle differences in meaning, since semantically speaking, the words *dog* and *cat* are also somewhat close in meaning, both characterizing some kind of animal or pet. We exemplify this in Table 4.1 for three representative W2W metrics, LSA (Landauer et al. 2007), Lin (Lin 1998) and HirstStOnge (Hirst and St-Onge 1998).

Table 4.1
Computing optimal word pairing for W2W similarity metrics

Word pairs	LSA	Lin	HirstStOnge
pet-pet	1	1	1
pet-cat	.508	.481	.2500
pet-dog	.479	.504	.2500
dog-cat	.140	.886	.1875
Average(pet-pet; dog-cat)	.5700	.9430	.5937
Average(pet-cat; pet-dog)	.4935	.4925	.2500

If there is no other option than to accept this issue of our current W2W metrics, then there is one option that one could use to improve the optimal matching, while still using these W2W metrics. The idea is to also look at the context of the words when matching them. The context is composed of other words surrounding the ones we are trying to match, or words which have syntactic relations with. For our example, if we try to pair the words *cat*, *dog* and *pet*, while looking at their corresponding pronouns, we would be more inclined to pair *(your) dog* with *(your) pet*, and (my) cat with *(my) pet* than the other way. In this current work, we do not further explore this option, leaving it for future research.

Our experiments showed that such examples, where optimal matching is needed, are very rare in practice, and in most cases, using a simple greedy approach for pairing similar words should be fairly sufficient. In the *greedy* approach, if L_1 is the list of words from first text, and L_2 is the list of words from the second text, then a word from L_1 should be paired with one word from L_2, with whom it has the maximum semantic similarity within all the other words in L_2 (according to the chosen W2W metric). For the *optimal matching* approach we need to solve the classic *assignment problem*, which can be solved in polynomial time using the *Hungarian Algorithm*. We give more details on this approach and present some experimental results with it in Section 4.7.

There is another concern that we need to be aware of, when using W2W similarity metrics. Most W2W metrics do not pay attention to the agreement factor between words, such as synonyms versus antonyms. We have previously raised this problem, in Section 1.3, when we defined a normalized metric of semantic similarity between words. For example, if we take the following two synonym words, *refuse* and *reject* and we calculate the similarity between them, based on the LSA metric, this gives us a similarity score of $.0730$, which is pretty low, considering that the words are actually synonyms in the WordNet database[1]. When comparing the word *reject* with its most common antonym, *accept*, then LSA tells us that the words are similar at $.4025$, on a normalized scale. This is contra-intuitive, and confirms the fact that LSA is in general less concerned about how two words agree on each other, and more attentive on how two words are being used in similar contexts. Similarly, for WordNet measures, none of those metrics operate on any kind of parameter that measures how much two words agree or disagree with each other.

4.2. WordNet Similarity

The word relatedness measures in WordNet Similarity rely on lexico-semantic information in WordNet to decide semantic similarity words. In WordNet, words that have the same meaning, i.e., synonyms, are grouped into synsets, or synonymous sets. For instance, the synset of *affectionate, fond, lovesome, tender,*

[1] Most WordNet-based measures will output a normalized similarity score of 1 between synonym words, like *refuse* and *reject*

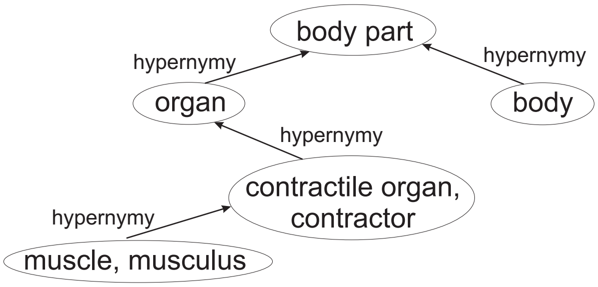


Figure 4.1
A snapshot of the WordNet taxonomy for nouns.

warm corresponds to the concept of *having or displaying warmth or affection*, which is the definition of the concept in WordNet. Each concept has attached to it a gloss: its definition and several usage examples. Words can belong to more than one synset (or concept), which is the case when they have more than one meaning. For example, the adjective *warm* belongs to 10 different concepts, each representing a different meaning of the word. In WordNet, concepts are linked via lexico-semantic relations such as hypernymy (is-a), hyponymy (reverse is-a), and meronymy (part-of). The nouns and verbs are organized into a hierarchy, or taxonomy, using the hypernymy relation. We show a snapshot of the WordNet hierarchy in Figure 1. WordNet contains only content words: nouns, verbs, adjectives, and adverbs.

WordNet has been used extensively to measure how similar two concepts are (Pedersen, Patwardhan, and Michelizzi 2004). In general, the underlying principle used in WordNet-based similarity measures is that two concepts are semantically more related if they are closer to each other in WordNet. For instance, the concept of *muscle, musculus* is more related to the concept of *contractile organ, contractor* than to *body* (see Figure 4.1). The reason is that the former two concepts are one *hypernymy* link away while the latter two are four links away (including one change of direction while following the *hypernymy* link between *body part* and *body*). The existing WordNet measures are usually divided into two groups: similarity measures and relatedness measures. The similarity measures are limited to concepts within the same taxonomy. That is, they can only compute similarity between words that are part of the same hierarchy of hypernymy relations (i.e., nouns or verbs). This limitation comes from the fact that similarity measures are based on the *is-a* relation, which is only used among nouns and among verbs in WordNet. As such, the similarity-based measures cannot be applied to adjectives and adverbs. The text relatedness measures are based on all types of relations in WordNet and thus can be used to compute similarity among words belonging to different category, e.g., between nouns and adjectives. Note that the way, in which the two concepts of similarity and relatedness are defined in WordNet, is similar to how we previously defined these relations in the introductory chapter of this dissertation (see Section 1.3). Therefore, we expect words that are considered to be close, according to some relation of semantic relatedness in WordNet, to convey similar meanings (i.e., *to simulate* vs. *simulation*) or at least be close in meaning (i.e., *happy* vs. *joy* or *excitement*).

4.2.1 WordNet Similarity Measures

In this subsection, based on the work of Corley and Mihalcea (2005), we detail six of the WordNet similarity measures that were available in the WordNet Similarity package (Pedersen, Patwardhan, and Michelizzi 2004) and that we experimented with. As previously mentioned, these measures can only be computed among content words which belong to the same taxonomy of concepts (e.g., noun vs. noun; or verb vs. verb).

Path length (path) is a simple counting scheme of nodes between two given concepts. This score is inversely proportional to the number of nodes along the shortest path between the synsets (see Equation 4.1, where *length* defines the shortest path). When comparing two words from the same synset, then the shortest possible path between them is considered to be 1. Therefore the maximum possible value for this metric is 1, and there is no further need to normalize it.

$$Sim_{path} = \frac{1}{length} \tag{4.1}$$

Leacock and Chodorow (lch) (Leacock and Chodorow 1998) extends the Path measure by applying a logarithm funtion, while also accounting for the depth of the taxonomy where the concepts are located. The maximum depth of the taxonomy is noted by D in Equation 4.2 of the lch measure. The maximum value of this measure depends on the depth of the taxonomy. For nouns we calculated this to be 3.6889, while for verbs the maximum is 3.3322. We normalize the lch measure by dividing it to the appropriate values.

$$Sim_{lch} = -log\frac{length}{2 * D} \tag{4.2}$$

Wu and Palmer (wup) (Wu and Palmer 1994) measure is based on the computed depths of the two concepts and the *least common subsumer* (LCS) between the concepts (see Equation 4.3). LCS represents the deepest node in the taxonomy that is a hypernym for both concepts. From this definition we notice that all possible values of the wup measure are between 0 and 1, so there is no need to normalize this further.

$$Sim_{wup} = \frac{2 * depth(LCS)}{depth(concept_1) + depth(contept_2)} \tag{4.3}$$

For the next three measures, we need to define the *information content* (IC) of a concept, as shown in Equation 4.4, where $P(c)$ is the probability of encountering an instance of concept c in a large corpus. The probability for the IC was pre-computed from various corpora (such as the British National Corpus, the Penn Treebank, the Brown Corpus, the complete works of Shakespeare, and SemCor) and provided along with the WordNet Similarity package.

$$IC(c) = -logP(c) \tag{4.4}$$

Resnik (res) (Resnik 1995) calculates the information content (IC) of the LCS of two contents (see Equation 4.5). This measure needs to be normalized in order to be used by our methods. We empirically found that the maximum IC for any concept used in our datasets is 11.76576, therefore we used this value to normalize the metric.

$$Sim_{res} = IC(LCS) \tag{4.5}$$

Lin (lin) (Lin 1998) measure builds on Resnik measure and adds a normalization factor based on the IC of the concepts that we compare (see Equation 4.6). There is no need to normalize this measure further.

$$Sim_{lin} = \frac{2 * IC(LCS)}{IC(concept_1) + IC(concept_2)} \tag{4.6}$$

Jiang and Conrath (jcn) (Jiang and Conrath 1997) is the last measure that we experiment with. Equation 4.7 shows how to compute the *jcn* measure, based on the IC and the LCS between two concepts. We found that sometimes this measure goes beyond the maximum normalized boundary of 1, and this happens for concepts which are so close in meaning that the denominator becomes less than 1. For such cases, we normalize the jcn values to a fixed value or 0.9999, since this is for input words that are really close in meaning but still not identical.

$$Sim_{jcn} = \frac{1}{IC(concept_1) + IC(concept_2) - 2 * IC(LCS)} \tag{4.7}$$

4.2.2 WordNet Relatedness Measures

The second group of WordNet metrics are called measures of relatedness (Pedersen, Patwardhan, and Michelizzi 2004), since they can be made across part of speech boundaries and are not limited to *is-a* relations. So they allow computing semantic similarities between concepts from different categories, or parts-of-speech (e.g., nouns vs. verbs). This cross-category feature provides a significant advantage over the metrics presented in previous subsection. For instance, looking back at the example provided in Section 1.6.2, the semantic similarity between the adjective *warmer* in the SE and the noun *heat* in the textbase T, can be computed with relatedness measures but not with similarity measures. The downside of these measures is that they are very slow, making it particularly difficult for us to experiment with them. The following word relatedness measures are available: HSO (Hirst and St-Onge 1998), LESK (Banerjee and Pedersen 2003), and VECTOR (Patwardhan 2003). Given two WordNet nodes, i.e., concepts, these measures provide a real value indicating how semantically related the two concepts are.

The HSO measure is path based, i.e., uses the relations between concepts, and assigns relations in WordNet a direction. For example, *is-a* relation is upwards, while *has-part* relation is horizontal. The relatedness between two concepts is derived by finding a path between them that is neither too long nor that changes direction too often. The LESK and VECTOR measures are gloss-based. That is, they use the text of the gloss as the source of meaning for the underlying concept. The LESK measure computes the overlap between the glosses of two concepts, as well as concepts that are directly linked to them in WordNet. The VECTOR measure creates a co-occurrence matrix from a corpus made up of WordNet glosses. Each content word used in a WordNet gloss has an associated context vector. Each gloss is represented by a gloss vector that is the average of all the context vectors of the words found in the gloss. Relatedness between concepts is measured by finding the cosine between a pair of gloss vectors.

Because of the complexity of these measures, which makes the process for computing them exceedingly slow, we found it difficult to experiment with them, on all of our datasets. Therefore, this dissertation contain only a few preliminary results (see Section 4.4) that we obtained, using these measure on the ULPC corpus (previously introduced in Section 1.6.2). Specifically, we present correlation numbers between these metrics and several dimensions of text-to-text semantic similarity relations that the ULPC corpus provides. Similar to our previous WordNet-based measures, the text relatedness measures are normalized by calculating the relatedness scores between any two words in the ULPC corpus (one in the textbase T and the other in the SE) and then taking the maximum. For instance, the HSO measure leads to a maximum similarity score of 16.

In agreement with what we said in the introductory part of this section, the W2W relatedness measures do not exploit the full potential of WordNet as for instance, *good* and *bad* are as similar as *bad* and *evil*. Using VECTOR as the measure, we get relatedness values of 0.7185 and 0.7142, respectively. This is simply the case because the W2W relatedness measures only account for the number and eventually direction of the links but not the label of the links. Between *good* and *bad* there is *antonymy* relation, while between *bad* and *evil* there is *similar-to* relation.

4.3. From Words to Concepts: Solving Word Sense Disambiguation

Before extending word-to-word (W2W) semantic similarity measures into text-to-text (T2T) semantic similarity measures, we need to clarify one issue that we are confronted with, particularly when using WordNet similarity measure, which is to convert a similarity between concepts into a similarity between words. In the case of LSA, the problem is simple, since each word is already assigned an LSA vector, and the cosine between two LSA vectors will give us the semantic similarity between their corresponding words. In case of WordNet similarity measure, the problem is a bit trickier. Given that texts express meaning using words and not concepts, we are faced with the challenge of mapping words from the input text to their corresponding concepts in WordNet. We are therefore faced with a word sense disambiguation (WSD) problem. It is beyond the scope of this work to fully solve the WSD problem, one of the hardest in the area of Natural Language Processing. Instead, we address the issue in two simple ways: (1) map the words from the input texts unto concepts corresponding to their most frequent sense, which is sense #1 in WordNet, and (2) map words onto all the concepts corresponding to all the senses and then take the maximum of all W2W relatedness scores for each pair of senses. In our experiments we label the decision of choosing one of these two ways, as the WSD decision and we label the first word sense disambiguation technique as ONE, i.e., sense one, whereas the latter is labeled as ALL, i.e., all senses of a word. Thus, the semantic similarity between two words is the W2W similarity between the concepts corresponding to the first senses of the words (for WSD option ONE) or the maximum over the W2W similarity scores obtained for all possible pairs of senses for the two words (for WSD option ALL).

4.4. Preliminary Results with WordNet Relatedness Measures on ULPC

This section describes a preliminary experiment with using WordNet relatedness measures to automatically assess student self-explanations in the intelligent tutoring system iStart, which we previously introduced in section 1.6.2. This approach is based on measuring the semantic similarity between a self-explanation (SE) and a reference text, called the textbase T (see Section 1.6.2 for a concrete example of T-SE pairs). The challenge is to decide, for instance, whether the SE is a paraphrase of the textbase T. Assessing the student self-explanations (SEs) is a critical step in iSTART because it is based on this assessment that the tutoring system could detect possible student misunderstandings and provide the necessary corrective feedback.

The semantic similarity is estimated using word relatedness measures, which rely on knowledge encoded in WordNet (Miller 1995). We also experimented with weighting words based on their importance, which is characterized by their corresponding IDF values computed from Wikipedia (see Section 3.9 for more details on these IDF values and how they were collected). The goal of this experiment was to answer the following two research questions: (1) are WordNet-based T2T relatedness measures competitive when compared to other approaches such as LSA and the Entailer and (2) is IDF-weighting important.

In addition, we compare the proposed algorithms to other approaches, namely LSA (Landauer et al. 2007) and the Entailer (Rus et al. 2008b). The LSA approach is based on representing texts into vectors in an LSA space (the LSA space dimensions, usually 300-500) which is automatically derived from large collection of texts using singular value decomposition (SVD), a technique for dimensionality reduction (see section 4.8.1 for more details about this). In Section 4.8 we work with a very similar method that uses LSA for vectorial representation of the input texts. The Entailer is an approach that relies on both lexical and syntactic information to detect text-to-text semantic relations among sentences. The Entailer proved to be quite successful according to several recent studies (Rus and Graesser 2007; Rus et al. 2008b).

4.4.1 Methods

We experimented with two methods of using W2W relatedness measures, depending on whether IDF weighting is being used or not. In the following, we denote with $wn_{rel}(v,w)$ a generic relatedness function between concepts v and w which would mean any of the three WordNet relatedness measures that were previously described in subsection 4.2.2.

Method 1. In this method, we extend the W2W measures to compute the relatedness between the textbase T and SE, i.e., to a T2T relatedness measure, by assessing how close the concepts in the textbase T are to the concepts the student articulates in the SE. A T-SE relatedness, $WN_{rel-score}(T,SE)$, is computed by taking the average of the best W2W relatedness scores between a textbase word and any word in the SE (see Equation 4.8). For words that have a direct match in the SE we assign the maximum relatedness score, which is 1 after normalization.

$$WN_{rel-score}(T,SE) = \frac{\sum_{v \in T} max_{w \in SE}\{wn_{rel}(v,w)\}}{|T|} \tag{4.8}$$

Method 2. This second method differs from the previous one in that each word in the textbase is weighted by its importance (see $WN_{IDF-rel-score}(T, SE)$ in Equation 4.9). The importance of a word is approximated using its specificity, or its IDF value. That is, the more specific a word is the most important that word is from the point of view of discovery semantic relations between texts. In other words, if a very specific term is mentioned in the textbase it should be mentioned directly or through a closely related concept in the SE.

$$WN_{IDF-rel-score}(T, SE) = \frac{\sum_{v \in T} idf(v) * max_{w \in SE}\{wn_{rel}(v, w)\}}{\sum_{v \in T} idf(v)} \tag{4.9}$$

4.4.2 Results

We used in our experiments the 1998 pairs of Textbase-SE in the ULPC (McCarthy and McNamara 2009). We evaluated the performance of the proposed methods along six of the ten dimensions of analysis available in the ULPC: elaboration, semantic completeness, entailment, lexical similarity, and paraphrase quality. As previously mentioned in subsection 1.6.2, dimensions such as *Garbage* are of not relevant for T2T relation detection. It should be noted that some of these dimensions, e.g., elaboration, semantic completeness, and paraphrase quality, have meanings that need be specified as they are not obvious or they differ from definitions used by others. Elaboration refers to SEs regarding the theme of the textbase rather than a restatement of the sentence. Semantic completeness refers to an SE having the same meaning as the textbase, regardless of word- or structural-overlap. Paraphrase quality takes into account semantic-overlap, syntactical variation, and writing quality. Given these definitions, the semantic completeness dimension in ULPC is equivalent of the paraphrase definition in the MSR corpus (Dolan, Quirk, and Brockett 2004).

Table 4.2
Correlations among Methods 1 and 2, human judgments, LSA, and Entailer.

Method	Elab	Sem-C	Ent	Lex-sim	Par-Q	W-Q
ONEHSO	-.156	.556	.515	.791	.318	.447
ONELESK	-.160	.550	.507	.784	.310	.441
ONEVECTOR	-.137	.567	.531	.800	.341	.495
ONEIDFHSO	-.171	.585	.539	.798	.387	.464
ONEIDFLESK	-.179	.576	.526	.792	.374	.456
ONEIDFVECTOR	-.149	.603	.563	.812	.422	.530
ALLHSO@	-.238*	.459	.422	.752	.081	.428
ALLLESK	-.146	.560	.517	.788	.323	.459
ALLVECTOR	-.110	.575	.541	.791	.362	.529
ALLIDFHSO@	-.233*	.476	.467	.734	.140!	.415
ALLIDFLESK	-.161	.583	.534	.792	.386	.473
ALLIDFVECTOR	-.113	.606	.568	.794	.443	.568
LSAassa	-.175	.555	.535	.804	.410	.498
R-Ent	-.177	.564	.512	.776	.321	.425
F-Ent	-.212	.449	.441	.726	.269	.405
A-Ent	-.204	.529	.497	.785	.308	.434

We have explored a space of 3x2x2=12 T2T solutions as a result of combining three relatedness measures (HSO, LESK, and VECTOR), two word sense disambiguation methods (ONE and ALL), and the two text-to-text relatedness methods (with and without IDF-weighting). We use the implementation of the HSO, LESK, and VECTOR measures from the WordNet::Similarity package (Pedersen, Patwardhan, and Michelizzi 2004). We looked first at correlations between the 12 solutions and human judgments. The correlation values are shown in Table 4.2 where the columns represent the five evaluation dimensions from ULPC mentioned above and the rows are different solutions. For instance, the row ALLIDFLESK means ALL the senses of words were used to compute relatedness between words, IDF weighting was active, and the relatedness measure used was LESK. When no IDF is mentioned it indicates no word weighting was used (Method 1). We also show in the table the correlation coefficients for LSA, and three variants of the Entailer, which were simply taken from the ULPC package, which includes the output for LSA and the Entailer on the dataset. We only picked these four external approaches because they were best correlated with human judgments (McCarthy and McNamara 2009) on the dimensions of interest for us.

Taking a quick glance at the results, we see that, in general, word weighting leads to better models compared to non-weighting models, correlation-wise. The same can be said about considering all senses of a word when computing W2W relatedness measures which, in most part, gives better correlation numbers than looking at only the most common sense of words.

4.5. Extending Lexical Methods with Word Semantics

In this section we present how to extend the lexical similarity methods described in previous chapter, with word-to-word semantic similarity measures that were introduced in the first part of this chapter.

For our previous methods, based on lexical token overlap, the core approach was to match lexical tokens from both input texts and then count them in order to measure how lexically similar the two texts are. Matching the lexical tokens was done on a simple binary, decision-based level, meaning two given tokens were either similar, and therefore could be matched, or not. Current chapter suggests that comparing and matching tokens could be done based on a normalized scale of similarity, as follows: given two input texts, A and B, for which we need to measure the degree of semantic similarity between them, and assuming that there is a metric, $\Re$, which measure the degree of similarity between any two given lexical tokens, then a token v from A should be matched with a token w from B that is the closest to v, according to the metric $\Re$. We call this the *greedy matching* approach. So far, this approach is similar to the ones described in Equations 4.8 and 4.9, in the previous section.

We now introduce two new conditions that need to be respected for a match to be possible. First, the two tokens, v and w, need not be already matched with other tokens. Second, the computed similarity between them needs to be above a certain threshold value, which we note with Th_{sim} (one default value for this Th_{sim} which proved to be good enough in our experiments is 0.5) . The first condition assures us that we are doing a 1-to-1 match of tokens between the two inputs, while second condition is needed to make sure that we are not matching tokens which are specific to only one text and should not be matched in the other (i.e., words such as *brief* in Text A and *team* in Text B, from the first example given in the introductory section of Chapter 1).

Once the two tokens v and w have been matched, we then compute their contribution to the total similarity score between A and B, as a direct proportional value to the degree of their similarity, based on the metric $\Re$. After all possible matches have been found and a final similarity score has been computed, we normalize this score as previously described in Chapter 3.7, by dividing it to some value that relates to the total number of tokens which are present in the input texts.

We now exemplify this general approach on the first example given in the introductory part of this chapter. After tokenization, ignoring the ending punctuation, and converting all lexical tokens to their base form, in lower case, we are left with the following pair of preprocessed sentences:

Text A: *york had no problem with mta s insist the decision to shift fund have been within its legal rights*

Text B: *york have no problem with mta s say the decision to shift fund be within its power*

Now, we pair all identical tokens into 15 perfect pairs, after which the following tokens are left unpaired:

Text A: *insist have legal rights*

Text B: *say power*

Next, we use the *lch* similarity measure from WordNet (Equation 4.2) to pair the two words from Text B with their closest words from Text A. We find that *say* is matched with *insist* with a similarity measure of $.67$ and *power* is matched with *rights* with a similarity score of $.51$. Since both similarity values are above our default Th_{sim} value of 0.5, we can accept both matches as pairs and calculate the final similarity score accordingly: $Sim_{total} = 15 + .067 + .51 = 16.18$. At last, we normalize this score, using *Max-Norm* (see Section 3.6), by dividing it to the total number of tokens from the longer Text A and we get a final, normalized similarity score of $Sim_{Max-Norm} = 16.18/19 = .852$ between the two initial inputs.

4.6. Performance Results on Greedy Methods based on Word Semantics

The methods that we experiment with in this section, we call them *greedy* because they are based on a *greedy matching* approach (as was described in previous section) of lexical tokens that are believed to be semantically close, according to some W2W similarity metric. As was previously suggested in introductory part of this chapter, there is another type of matching that can be done, which is to find the *optimal matching* between the tokens of the two inputs, according to some given W2W metric. We shall study this other type of matching in the section which follows after this.

Again, we are faced with a similar problem as in the previous chapter, which is that there are too many options to explore. In previous Section 3.8 we noted that there are 288 possible combinations of lexical methods to explore, for one dataset, one normalization and one type of preprocessing. Now we have two more parameters that we need to consider: a) a total of 7 W2W similarity metrics (i.e., *path, lch, wup, res, lin, jcn* of WordNet-based similarity measures, and LSA); and b) two WSD methods to map words into concepts, when using the WordNet-based W2W measures (i.e., WSD_{one} and WSD_{all}). This multiplies our initial search space by almost 14 times. Therefore we decided to select only a few of the lexical-based methods that were previously experimented with, and extend them with the W2W similarity metrics. We did an empirical analysis of the data, and found out that, in general, those lexical-based methods which offered best *precision* scores on the test data, report an improvement in performance, both in terms of accuracy and precision, when enhanced with W2W semantic measures.

To conclude, we selected best combination of lexical-based methods, in terms of precision on the test data, and evaluate them on both types of preprocessing, OpenNLP and Stanford. We also experimented with altering the default Th_{sim} value for accepting a found match as a similarity pair. In general we found that a value of 0.5 for Th_{sim} works best in most cases, and so we used this values in all experiments presented in this chapter.

Another detail that we must consider for our experiments is on how to compare the base form of the words. For WordNet similarity metrics, the solution is already solved, since WordNet considers only the lemma form of words when looking at their corresponding concepts in its database. So the only difference when comparing the initial raw forms of words versus their base forms is when we are initially looking for perfectly identical pairs of tokens, that is before matching the remained unpaired tokens based on a selected semantic similarity measure. For LSA however, the words are encoded into the LSA space in their original raw form. Therefore we had to consider all words in the dataset that converge to the same base form, take their corresponding LSA vectors, and combine then into a single vector that will reflect the meaning of their base form (by combining, we mean do a weighted sum of these vectors - the weight is based on a precomputed entropy of the participating words - and then normalize the output vector).

Tables 4.3 and 4.4 present performance scores on the MSR corpus, when using OpenNLP and Stanford preprocessing, accordingly. The format of these tables follows the same format as in Section 3.8. We experimented with both WSD options for the WordNet based metrics (marked with WSD_{one} for choosing the most common sense of a word, and WSD_{all} for looking at all senses of a word).

When comparing between the two WSD methods we see that using all senses of a word gives better accuracy results on the testing data, although there is a better precision when using WSD_{one} option, with the *lch* metric. When looking at the different metrics used, we see that the *lch* measure gives us better scores than the other measures, in terms of accuracy and precision. Also using Stanford preprocessing seems to work better than OpenNLP when including word semantic measures. We see that a WSD_{all} approach, with *lch*, gives us best accuracy on test, obtained so far on MSR - $.7559$, while a WSD_{one} approach, with *lch* again, gives us best precision on test, obtained so far on MSR - $.7942$.

Another observation that we can make here is that the optimum thresholds slightly increased, when compared to the lexical based methods. This is because

Table 4.3
W2W Semantic methods on MSR with OpenNLP (`P.B.C.U.N.N.` method, with Average-Norm and greedy matching)

		Performance on Train			Performance on Test		
Method	Threshold	Acc.	Prec.	Recall	Acc.	Prec.	Recall
WSD_{one} - path	.5957	.7301	.7730	.8500	.7264	.7698	.8396
WSD_{one} - lch	.6111	.7341	.7744	.8554	.7333	**.7707**	.8527
WSD_{one} - wup	.5964	.7343	.7699	.8652	.7293	.7632	.8596
WSD_{one} - res	.6055	.7252	.7456	.9005	.7165	.7330	**.9024**
WSD_{one} - lin	.6000	.7323	.7730	.8547	.7281	.7703	.8422
WSD_{one} - jcn	.5957	.7306	.7734	.8503	.7264	.7698	.8396
WSD_{all} - path	.5957	.7309	.7717	.8543	.7316	.7706	.8492
WSD_{all} - lch	.6140	.7392	.7677	.8801	**.7438**	.7677	.8814
WSD_{all} - wup	.6110	.7363	.7630	.8841	.7362	.7559	.8910
WSD_{all} - res	.5978	.7336	.7716	.8602	.7339	.7679	.8596
WSD_{all} - lin	.5990	.7353	.7678	.8718	.7339	.7658	.8640
WSD_{all} - jcn	.5957	.7306	.7709	.8554	.7316	.7706	.8492
LSA	.5990	.7309	.7687	.8605	.7304	.7668	.8544

Table 4.4
W2W Semantic methods on MSR with Stanford, (`W.B.I.U.N.F.` method, with Max-Norm and greedy matching)

		Performance on Train			Performance on Test		
Method	Threshold	Acc.	Prec.	Recall	Acc.	Prec.	Recall
WSD_{one} - path	.5263	.7321	.7775	.8453	.7351	.7764	.8448
WSD_{one} - lch	.5544	.7387	.7963	.8238	.7362	**.7942**	.8143
WSD_{one} - wup	.5245	.7343	.7701	.8649	.7357	.7676	.8640
WSD_{one} - res	.5325	.7274	.7465	.9030	.7258	.7393	.9076
WSD_{one} - lin	.5244	.7345	.7762	.8529	.7357	.7740	.8509
WSD_{one} - jcn	.5263	.7323	.7776	.8456	.7351	.7764	.8448
WSD_{all} - path	.5000	.7336	.7510	.9059	.7484	.7563	.9172
WSD_{all} - lch	.5507	.7424	.7806	.8605	**.7559**	.7895	.8631
WSD_{all} - wup	.5517	.7392	.7765	.8620	.7490	.7802	.8666
WSD_{all} - res	.5309	.7355	.7802	.8471	.7420	.7831	.8466
WSD_{all} - lin	.5219	.7368	.7675	.8754	.7449	.7688	.8814
WSD_{all} - jcn	.5000	.7350	.7513	.9085	.7472	.7548	**.9180**
LSA	.5238	.7328	.7708	.8602	.7374	.7711	.8605

Table 4.5
W2W Semantic methods on ULPC with OpenNLP, (`W.W.I.U.N.N.` method, with Average-Norm and greedy matching)

		Performance on Train			Performance on Test		
Method	Threshold	Acc.	Prec.	Recall	Acc.	Prec.	Recall
WSD_{all} - path	.3846	.6451	.6219	.8632	.6232	.6164	.8242
WSD_{all} - lch	.3478	.6444	.6106	.9303	**.6533**	.6208	**.9414**
WSD_{all} - wup	.4224	.6444	.6204	.8682	.6313	.6180	.8535
WSD_{all} - res	.3571	.6451	.6166	.8943	.6353	.6176	.8755
WSD_{all} - lin	.3828	.6451	.6185	.8831	.6293	.6158	.8571
WSD_{all} - jcn	.3750	.6458	.6201	.8769	.6253	.6150	.8425
LSA	.3831	.6391	.6175	.8595	.6353	**.6220**	.8498

when using word semantics the algorithm tends to find more pairs, since we are also matching words that do not have an exactly identical lexical form in the other input text.

For the next two datasets, ULPC and RTE, we show results only on the WSD_{all} approach, since this one seemed to perform better on MSR, than the other approach. Tables 4.5 and 4.6 present performance results on these other two datasets, ULPC and RTE respectively, both on the same combination of preprocessing options (compare only words, in their original form, case insensitive, and no weighting).

Although on ULPC, by using word semantics, we managed to raise the performance bar a little, in terms of all three measures of accuracy ($.6533$ with *lch*), precision ($.6220$ with *LSA*) and recall ($.9414$ with *lch*), we noticed that the RTE corpus was not so friendly with these metrics. In this case we barely managed to get a slightly better precision when using the *path* measure ($.6269$ when compared to $.6269$, from the previous lexical-based methods).

Table 4.6
W2W Semantic methods on RTE with Stanford, (`W.W.I.U.N.N.` method, with asymmetric normalization, $A \rightarrow B$ and greedy matching)

		Performance on Train			Performance on Test		
Method	Threshold	Acc.	Prec.	Recall	Acc.	Prec.	Recall
WSD_{all} - path	.6154	.6018	.5889	.6902	**.6313**	**.6269**	.6927
WSD_{all} - lch	.6130	.6042	.5757	.8106	.6162	.5938	**.7951**
WSD_{all} - wup	.6667	.5893	.5722	.7277	.6175	.6028	.7439
WSD_{all} - res	.6154	.6007	.5854	.7071	**.6313**	.6226	.7122
WSD_{all} - lin	.6105	.5968	.5793	.7251	.6212	.6090	.7293
WSD_{all} - jcn	.6154	.6015	.5882	.6934	.6300	.6239	.7000
LSA	.6250	.6037	.5896	.6982	.6313	.6226	.7122

4.7. Performance Results on Optimal Methods based on Word Semantics

As previously mentioned in the introductory part of this chapter, by introducing the concept of words semantics in the problem of textual similarity assessment, it allows for another type of matching the lexical tokens between two input texts, and we call this the *optimal matching* approach. Optimal matching algorithms try to find optimal solutions of pairing elements from two distinct sets, such that it maximizes a particular function that depends directly on the similarity scores between the paired elements. In the introductory section of this chapter we gave one example where such an optimal matching might prove useful for our problem, given that an efficient W2W semantic similarity metric is available to us.

One major issue with problems that require finding optimal solutions is time complexity. For our problem we observe that it closely related to the classic problem of optimal assignment. The *assignment problem* is one of the fundamental combinatorial optimization problems where the task is to find a maximum weight matching in a weighted bipartite graph. A good definition of this problem can be found in Wikipedia, where in its most general form, the problem is as follows:

> **The Task Assignment Problem.** There are a number of *agents* and a number of *tasks*. Any agent can be assigned to perform any task, incurring some *cost* that may vary depending on the agent-task assignment. It is required to perform all tasks by assigning exactly one agent to each task in such a way that the *total cost* of the assignment is minimized.[2]

The difference between this and our problem is that the agents are words from the first sentence, the tasks are words from the second sentence (these can be interchanged, in the case of symmetric similarity), the cost function is given by the similarity metric between words, and we are instead looking to find the maximum *total cost*, instead of the minimum.

One efficient way to solve the assignment problem in *polynomial time* is given by the **Hungarian method**, which is a combinatorial optimization method, developed and published by Harold Kuhn in 1955 (Kuhn 2005) and named after the earlier works of two Hungarian mathematicians, Konig and Egervary. We used an existing java implementation of the algorithm[3] which needs as input a matrix of computed W2W similarity measures for all the possible combination of pairs between the two input texts, and outputs a combination of pairing that provides maximum similarity-based matching. On our experiments, since the input texts were not too long, computing this matrix didn’t seem very demanding for our system.

[2] Taken verbatim from the Wikipedia page on the *Assignment problem* - June, 2011

[3] Created and made available by Konstantinos A. Nedas from University of Maine, Orono, ME

Table 4.7
W2W Semantic methods on MSR with optimal matching

		Performance on Train			Performance on Test		
Method	Threshold	Acc.	Prec.	Recall	Acc.	Prec.	Recall
WSD_{all} - path	.5469	.7318	.7652	.8700	.7391	.7699	.8666
WSD_{all} - lch	.5959	.7289	.7661	.8616	.7304	.7648	.8588
WSD_{all} - wup	.5862	.7309	.7475	.9085	.7270	.7421	**.9032**
WSD_{all} - res	.5731	.7301	.7853	.8264	.7299	**.7821**	.8230
WSD_{all} - lin	.5728	.7318	.7756	.8485	**.7403**	.7816	.8457
WSD_{all} - jcn	.5500	.7316	.7706	.8580	.7345	.7710	.8544
LSA	.5385	.7257	.7480	.8958	.7310	.7517	.8893

Table 4.8
W2W Semantic methods on ULPC with optimal matching

		Performance on Train			Performance on Test		
Method	Threshold	Acc.	Prec.	Recall	Acc.	Prec.	Recall
WSD_{all} - path	.3542	.6618	.6272	.9104	.6493	.6276	**.8828**
WSD_{all} - lch	.4459	.6531	.6282	.8657	.6333	.6250	.8242
WSD_{all} - wup	.4474	.6518	.6233	.8868	.6353	.6233	.8425
WSD_{all} - res	.3720	.6571	.6259	.8968	.6413	.6243	.8645
WSD_{all} - lin	.3790	.6598	.6261	.9080	.6373	.6211	.8645
WSD_{all} - jcn	.3684	.6624	.6309	.8930	**.6513**	**.6341**	.8571
LSA	.3620	.6578	.6238	.9117	.6353	.6170	.8791

In the followings, we evaluated the *optimal matching* approach on the MSR and ULPC datasets. We used Stanford preprocessing, a Max-Norm option for normalization and the following combination of preprocessing options (`P.B.I.U.N.F` - include punctuation, use base form, and a local weight on the frequency of tokens). Tables 4.7 and 4.8 report performance scores for the two datasets respectively.

When comparing these results with the ones based on the greedy matching approach (Tables 4.4 and 4.5) we see that they do not show that much promise.

The only improvement that we see for the testing part of the corpora is on the ULPC data where, for the *jcn* metric, we managed to get a precision score of $.6341$, the highest obtained so far on this corpus. This partly reinforces our concern that using current W2W metrics on isolated tokens only, does not exploit the full potential of the optimal matching approach. Of course these results tell only a small part of the whole story, since there are plenty of other options to experiment with; but we will leave these for future work.

4.8. Latent Semantic Analysis

Another different approach to compute text-to-text similarity is by extending known statistical techniques for representing the meaning of words with vectorial representations. We mention here two such techniques, both of which are conceptually very similar. One is called Latent Dirichlet Allocation (LDA, (Blei, Ng, and Jordan 2003)), which is based on topic analysis, and the other is called Latent Semantic Analysis (LSA, (Landauer et al. 2007)), which is based on concept analysis. Although LSA is more commonly used when compared to LDA, one of the benefits of topic analysis over concept analysis is the ability to better handle polysemy and synonymy. In this section we're going to focus on studying LSA, also known as Latent Semantic Indexing. LSA had been previously described in section 2.4.2 where it was used for a semantic representation of texts. We leave the study of LDA for future research.

So far in this chapter we used LSA as one of several ways to measure the semantic similarity between words, which we employed to found matches of pairs between two input texts. What we are going to study in this section is how to extend the vectorial representations on the meaning of all the words in a text into a single vectorial representation for the whole text. Specifically, we are going to investigate the impact of several local and global weighting schemes on Latent Semantic Analysis' ability to capture semantic similarity between two texts. We worked with texts varying in size from sentences to paragraphs. We present a comparison of 3 local and 3 global weighting schemes across 3 different standardized data sets related to semantic similarity tasks. For local weighting, we used binary weighting, term-frequency, and log-type. For global weighting, we relied on binary, inverted document frequencies (IDF) collected from the English Wikipedia, and entropy, which is the standard weighting scheme used by most LSA-based applications. We studied all possible combinations of these weighting schemes on the following three tasks and corresponding data sets: paraphrase identification at sentence level using the MSR corpus (see Section 1.6.1) , paraphrase identification at sentence level using data from the intelligent tutoring system iSTART (which is encoded in the ULPC corpus - Section 1.6.2), and mental model detection based on student-articulated paragraphs in MetaTutor (Azevedo et al. 2008), another intelligent tutoring system. Our experiments revealed that for sentence-level texts a combination of type frequency local weighting in combination with either IDF or binary global weighting works best. For paragraph-level texts, a log-type local weighting in combination with binary global weighting works best. We also found that global weights have a greater impact for sentence-level similarity as the local weight is undermined by the small size of such texts.

4.8.1 Weighting in LSA

LSA represents the meaning of individual words using vector-based representations. The similarity of two words can be computed as the normalized dot-product between their corresponding vectors. In Section 2.4.2 we described the general framework to construct these LSA representations from sizable collections of documents. During this process, several weighting schemes can be used to improve the output.

As the LSA representation relies on word co-occurrences in the given collection of documents, various techniques have been used to reduce the role of highly-frequent words, e.g., *the*, which do not carry much meaning (Dumais 1991; Berry, Dumais, and O'Brien 1995; Nakov, Popova, and Mateev 2001). These techniques combine local weighting, quantifying the importance of words within the texts, and global weighting, which quantifies the importance of words in a large corpus, i.e., across many texts. Comparison among several weighting schemes for deriving the LSA-based representations of individual words have been done before and the choice of local and global weighting have been shown to have a significant impact on the overall performance of LSA-based semantic similarity methods (Landauer et al. 2007). For instance, Dumais (1991) has experimented with six weighting schemes that were derived from combining 3 local and 4 global schemes (not all combinations were explored). The most successful combination was based on the log of the local (within a document) term frequency and the inverse of the entropy of the term in the corpus. Nakov and colleagues (2001) experimented with 12 combinations of more or less the same set of local and global weights and found similar results, i.e., a combination of log and entropy is best. It is important to note that Dumais and Nakov and colleagues focused on different tasks: information retrieval and text classification, respectively, which is different than our task on text-to-text similarity assessment, at sentence and paragraph level.

But a more important distinction from previous work is that we investigate a different type of weighting for LSA, which becomes apparent when one tries to extend the LSA-based vectorial representation to assess the similarity of texts beyond word-level. We therefore experiment with weighting schemes to extend the LSA representation to sentences and paragraphs, after the derivation of the LSA representation of individual words. To the best of our knowledge, the role of this type of weighting has not been investigated before. Previous research on local and global weighting schemes for LSA has focused on weighting before the derivation of the LSA representation, i.e., during the creation of the term-by-document frequency matrix which is the input to the LSA procedure to derive the LSA representation for words. The term-by-document matrix contains information about which word occurred in which document in a large collection of documents.

Furthermore, our study has been extensively conducted across various types of data input to assess the role of local and global weighting schemas for the problem of textual similarity assessment. It is important to assess the role of weighting schemes across texts of various sizes as some weights may behave differently depending on the size of the text. For instance, the local weight of raw frequency which counts the number of occurrences of a word in the text will be dominated by the global weight in texts the size of a sentence because in such texts raw frequency is most of the time 1. This is explained by the fact that words are not reused in a sentence while in a paragraph they are, e.g., for cohesion purposes.

4.8.2 LSA-based Similarity of Texts

To compute how similar two words based on LSA vector representations, the cosine between the vectors must be computed. The cosine is the normalized dot product between the vectors. If we denote $V(w)$ the LSA vector of a word w then the cosine is given by Equation 4.10.

$$LSA(w_1, w_2) = \frac{V(w_1) * V(w_2)}{||V(w_1)|| * ||V(w_2)||} \tag{4.10}$$

A cosine value of 0 means there is no semantic similarity between words or paragraphs while 1 means they are semantically equivalent (synonyms).

The use of LSA to compute similarity of texts beyond word-level relies mainly on combining the vector representation of individual words. Specifically, the vector representation of a text containing two or more words is the weighted sum of the LSA vectors of the individual words. If we denote $weight_w$ the weight of a word as given by some scheme, local or global, then the vector of a text T (sentence or paragraph) is given by Equation 4.11. In Equation 4.11, w takes value from the set of unique words in text T, i.e., from the set of word types of T. If a word type occurs several times in a document that will be captured by the local weight ($loc - weight$). $Glob - weight$ in Equation 4.11 represents the global weight associated with type w, as derived from a large corpus of documents.

$$V(T) = \sum_{w \in T} \text{loc-weight}_w * \text{glob-weight}_w * V_w \tag{4.11}$$

To find out the LSA similarity score between two texts $T1$ and $T2$, i.e., $LSA(T1, T2)$, we first represent each sentence as vectors in the LSA space, $V(T1)$ and $V(T2)$, and then compute the cosine between two vectors, in a similar way as for the word level (see Equation 4.10).

There are several ways to compute local and global weights, of which some were already detailed in Section 3.4. For local weighting, the common schemes we looked at are: *binary*, *type frequency* and *log-type frequency*. *Binary* means 1 if the word type occurs at least once in the document and 0 if it does not occur at all. *Type frequency* weight is defined as the number of times a word type appears in a text, sentence or paragraph in our case. *Log-type frequency* weight is defined as log(1 + *type frequency*). It has been proposed by (Dumais 1991) based on the observation that type frequency gives too much weight/importance to very common, i.e., frequent, words. A frequent word such as *the* which does not carry much meaning will have a big impact, although its entropy (see Section 3.4) is low, which is counterintuitive. To diminish 'the frequency factor for such words, but not eliminate it entirely, the log-type weighting scheme was proposed.

For global weighting, we looked at: *binary* weight (similarly to local binary weight previously described), *entropy* weight (previously detailed in Section 3.4), and an *IDF*-based weight.

Given the need for word distributional information in our global weighting schemes, i.e., entropy and IDF, it is important to derive as accurate estimates of word statistics as possible. Accurate word statistics means being representative of overall word usage (by all people at all times). The accuracy of the estimates is largely influenced by the collection of texts where the statistics are derived from. Various collections were used so far to derive word statistics. For instance, (Corley and Mihalcea 2005) used the British National Corpus as a source for their IDF values. We chose Wikipedia instead, to derive the IDF index, because it encompasses texts related to both general knowledge and specialized domains and it is being edited by many individuals, thus capturing diversity of language expression across individuals. Furthermore, Wikipedia is one of the largest publicly available collections of English texts. Extracting IDF values and word statistics from very large collections of text, such as Wikipedia, is non-trivial task.

Section 3.9 gives ample details on our own implementation for extracting these values.

4.8.3 Prior Knowledge Activation Paragraphs

Before going into experimental part of this study, we must first introduce a new corpus, the PKA corpus, which we will be experimenting with in this study, along with the two other corpora that were previously introduced in the introductory chapter of this dissertation, the MSR (see Section 1.6.1) and the ULPC (see Section 1.6.2) datasets.

This new third corpus is a bit different than the previous two, in the sense that it contains paragraphs which must be labeled in three classes depending on their level of similarity to another paragraph of reference. The corpus was created to help evaluating methods that classify textual inputs given by students in the Intelligent Tutoring System, MetaTutor (Azevedo et al. 2008), which was briefly introduced in Section 1.7. At the beginning of their interaction with MetaTutor, students are given a learning goal, e.g., *learn about the human circulatory system*, and encouraged to use a number of self-regulatory processes that will eventually help with their learning. One of the important self-regulatory processes in MetaTutor is prior knowledge activation (PKA), which involves students recalling knowledge about the topic to be learned. During prior knowledge activation, students must write a paragraph which is assumed to reflect students knowledge with respect to the learning goal.

The corpus contains 309 paragraphs composed by students during PKA. The PKA paragraphs given by students are assumed to reflect students knowledge about the current topic, in other words, the students current mental model. A proper automatic evaluation of these paragraphs will help an interactive tutoring system to evaluate the student, measure its learning, and give feedback or act depending on students current level of mental knowledge. The paragraphs in the corpus are labeled by human experts on three levels of mental models (MM): High (100 paragraphs), Medium (70 paragraphs) and Low (139 paragraphs). For this corpus we compare each student paragraph with one ideal paragraph which is considered as benchmark for a perfect mental model, representing the highest level of MM. This ideal paragraph was created by a human expert and contains summarized information about all important concepts encompassed in the learning topic. An example of a student paragraph and its corresponding ideal paragraph, was previously given in Table 1.1 of Section 1.7.

4.8.4 Experiments and Results

We have explored all 9 possible combinations among local and global weighting schemes on three different datasets: Microsoft Paraphrase Corpus (MSR corpus), iSTART/ULPC, and PKA/MetaTutor. For each dataset, the task was to compute similarity between two texts and assess how well the LSA based predictions matched expert judgments. In the MSR and iSTART corpora, texts are the lengths of a sentence while the PKA data set contains texts the size of paragraphs. Details about each dataset will be provided in the next subsections.

We calculate LSA-based similarity between pairs of texts using all combinations of weighting schemes presented earlier and use logistic regression from WEKA machine learning toolkit (Witten and Frank 2005) to classify instances based on the LSA score. We report results using five performance metrics: accuracy, kappa measure, and weighted averages for precision, recall and f-measure. Accuracy is computed as the percentage of a method's prediction that matches the expected predictions suggested by experts. Kappa coefficient measures the level of agreement between predicted categories and expert-assigned categories while also accounting for chance agreement. Precision and Recall have been previously defined in first part of Section 3.8 for cases where there are only two classes to consider. For the PKA dataset, where there are multiple classes to consider (i.e., low, medium and high MMs), precision and recall are computed as averages per class. F-measure is calculated as the harmonic mean of precision and recall. Results were obtained using 10-fold cross-validation, except for MSR dataset for which we used the explicit test subset.

Similar as before, we used an LSA space from the TASA corpus. The corpus contains a distribution of texts from all genres (e.g., science, language arts, health, economics, social studies, business, and others) and has been successfully used as to derive LSA spaces by others (Graesser et al. 2007). For a good approximation on the meaning of words, it is usually recommended the LSA space to have between 100 and 500 dimensions (Landauer et al. 2007), which are viewed as relevant abstract or latent concepts to which a given word is more or less related to. In our case here, the LSA space has 326 dimensions.

We present three tables; each table corresponds to one dataset. Lines are specific to global weighting schemes, and on columns are listed each of the five evaluation measures grouped by local weighting schemes. We list all possible combinations of three global weighting schemes (binary weighting, entropy weighting, and IDF weighting) with three local weighting schemes (binary weighting, type frequency weighting, and log-type frequency weighting).

Table 4.9 presents results on the MSR corpus, Table 4.10 reports results on the ULPC corpus, while Table 4.11 shows results on the PKA corpus.

To synthesize these results, the next Table 4.12 presents combinations of local and global weighting schemes on which best results were reported, in terms of

Table 4.9
LSA results on the MSR dataset.

	binary (local)			type freq. (local)			log-type freq. (local)		
(global)	Acc	Prec	Rec.	Acc	Prec	Rec.	Acc	Prec	Rec.
binary	70.38	.686	.704	70.55	.689	.706	70.20	.684	.702
entropy	69.16	.669	.692	68.98	.667	.690	69.22	.670	.692
idf	69.85	.679	.699	69.74	.667	.697	69.85	.679	.699

Table 4.10
LSA results on the iSTART/ULPC dataset.

	binary (local)			type freq. (local)			log-type freq. (local)		
(global)	Acc	Prec	Rec.	Acc	Prec	Rec.	Acc	Prec	Rec.
binary	61.21	.618	.612	61.11	.616	.611	61.66	.624	.617
entropy	62.61	.630	.630	62.61	.631	.626	62.66	.632	.627
idf	63.21	.638	.632	63.21	.639	.632	63.16	.638	.632

Table 4.11
LSA results on the MetaTutor/PKA dataset

	binary (local)			type freq. (local)			log-type freq. (local)		
(global)	Acc	Prec	Rec.	Acc	Prec	Rec.	Acc	Prec	Rec.
binary	60.84	.472	.608	58.58	.456	.586	61.16	.473	.612
entropy	58.25	.450	.583	58.25	.451	.583	59.55	.461	.595
idf	60.19	.465	.602	59.87	.463	.599	59.55	.461	.595

Table 4.12
Weighting scheme combinations corresponding to best results for each dataset.

	ULPC	MSR	PKA
global×local	idf×type	bin×type	bin×log-type
accuracy	63.21	70.55	61.16
kappa	.239	.244	.354
precision	.639	.689	.473
recall	.632	.706	.612
f-measure	.616	.672	.534

accuracy, for each dataset. For sentence-level texts a combination of type frequency local weighting in combination with either IDF or binary global weighting works best. For paragraph-level texts, a log-type local weighting in combination with binary global weighting works best. From the table, we can see that there is no clear winner of global and local weight combination across tasks and text sizes. That may be a result of different distribution of positive and negative examples in the three data sets and that on MSR we used a training-test form of evaluation while for the other we used 10-fold cross-validation.

We also conducted a repeated measures analysis of variance with the local weighting as a repeated measurement. The differences among the various local weightings were significantly different at $p<0.05$ with the exception of raw and binary local weighting for sentence-level texts. This could be explained by the fact that for such texts the raw frequency and the binary value for binary weighting coincides simply because words tend to occur only once in a sentence. That is, words are less likely to be re-used in a sentence as opposed to a paragraph.

These experiments on LSA revealed that for sentence-level texts a combination of binary local weighting in combination with either IDF or binary global weighting works best. For paragraph-level texts, a log-type local weighting in combination with binary global weighting works best. We also found that global weights have a greater impact for sentence-level similarity as the local weight is undermined by the small size of such texts. Furthermore, we conclude from these experiments that there is no clear winning combination of global and local weighting across tasks and text size, which is somehow different from earlier conclusions for different types of weighting in LSA, at word-level representations, that entropy and log-type is the best combination. An idea for future research on this particular topic, would be to further explore the role of local and global weighting in more controlled experiments in which the same distributions of positive and negative examples is used, and same evaluation methodology, 10-fold cross-validation, across all data sets and text sizes.

Chapter 5

Syntactic Dependency Relations

In this chapter we go beyond simple word based matching, for the task of semantic similarity assessment, and study the syntactic relationship between words while looking for similarities and differences between two texts. We focus this chapter on an earlier study that we did on the MSR corpus to use dependency relations to measure semantic similarity between texts, at sentence level. This was a novel, fully automated approach to the task of paraphrase identification. The proposed approach quantifies both the similarity and dissimilarity between two sentences, mainly based on syntactic dependency relations between words. The basic idea is that two sentences are in a paraphrase relation if they have many similarities (at lexico-semantic and syntactic levels) and few or no dissimilarities. For instance, the two sentences in the MSR example shown in the beginning of previous chapter (Chapter 4) have many similarities, e.g., common words such as *York* and common syntactic relations such as the *subject* relationship between *York* and *have*, and only a few dissimilarities, e.g., Text A contains the word *saying* while Text B contains the word *insisting*. Thus, we can confidently deem the two sentences as being paraphrases of each other. Following this basic idea, to identify paraphrases we first compute two scores: one reflecting the similarity and the other the dissimilarity between the two sentences. A paraphrase score is generated by taking the ratio of the similarity and dissimilarity scores. If the ratio is above a certain threshold, the two sentences are judged as being paraphrases of each other. Similar to previous methods, we find the best separating threshold, by optimizing the performance of the proposed approach on training data. One important difference in this current approach is that the paraphrase score is not a normalized value, and so the threshold value is not bounded between the values of 0 and 1.

5.1. Distinctions from Previous Work

There are several key features of this approach that distinguish it from other approaches (see future Section 7.1) on measuring semantic similarity between texts, particularly for the task of paraphrase identification. *First*, it considers both similarities and dissimilarities between sentences. This is an advantage over approaches that only consider the degree of similarity (Rus et al. 2008a) because the dissimilarity of two sentences can be very important to identifying paraphrasing, as shown by the work of (Qiu, Kan, and Chua 2006), which is detailed later, in Section 7.1 of this dissertation.

Second, the similarity between sentences is computed using word-to-word similarity metrics instead of simple word matching or synonymy information in a thesaurus as in (Qiu, Kan, and Chua 2006; Rus et al. 2008a). The word-to-word similarity metrics can identify semantically related words even if the words are not identical or synonyms. We use the similarity metrics from the WordNet similarity package (Pedersen, Patwardhan, and Michelizzi 2004) that we've detailed in previous chapter, section 4. Similarly, Corley and Mihalcea (2005), used the WordNet similarity package to obtain semantic similarity between individual words. Our approach has the advantage that it also considers syntactic information, in addition to word semantics, to identify paraphrases.

Third, we weight dependencies to compute dissimilarities between sentences as opposed to simple dependency overlap methods that do no weighting (Lintean, Rus, and Graesser 2008; Rus et al. 2008a). Using dependencies provides an advantage over approaches that use phrase-based parsing (Rus et al. 2008a), since the latter limits the applicability of the approach to free-order languages, for which dependency parsing is more suitable. The weighting allows us to make fine distinctions between sentences with a high similarity score that are paraphrases and those that are not due to the strength of the few dissimilarities. For instance, two sentences that are almost identical except their subject relations are likely to be non-paraphrases as opposed to two highly similar sentences that differ in terms of, say, determiner relations. We weight dependencies using two features: (1) the type/label of the dependency, and (2) the depth of a dependency in the dependency tree. To extract dependency information, two parsers were used, Minipar (Lin 1993) and the Stanford parser (de Marneffe, MacCartney, and Manning 1993). We report results with each of the parsers later in the chapter.

A final feature that makes this approach particularly interesting from other approaches is that it is minimally-supervised. For example Zhang and Patrick (2005) used decision trees to classify the sentence pairs making their approach a supervised one. In one particular case we only need to derive the value of the threshold from training data for which it is only necessary to know the distribution of true-false paraphrases in the training corpus and not the individual judgment for every instance in the corpus. Also, Zhang and Patrick's approach relies only on lexical and syntactic features while we also use semantic similarity factors.

5.2. Approach

As mentioned before, our approach is based on the observation that two sentences express the same meaning, i.e., are paraphrases, if they have all or many words and syntactic relations in common. Furthermore, the two sentences should have few or no dissimilar words or syntactic relations. In the example below, we show two sentences with high lexical and syntactic overlap. The different information, *legal rights* in the first sentence and *powers* in the second sentence, does not have a significant impact on the overall decision that the two sentences are paraphrases, which can be drawn based on the high degree of lexical and syntactic overlap.

Text A: *The decision was within its legal rights.*

Text B: *The decision was within its powers.*

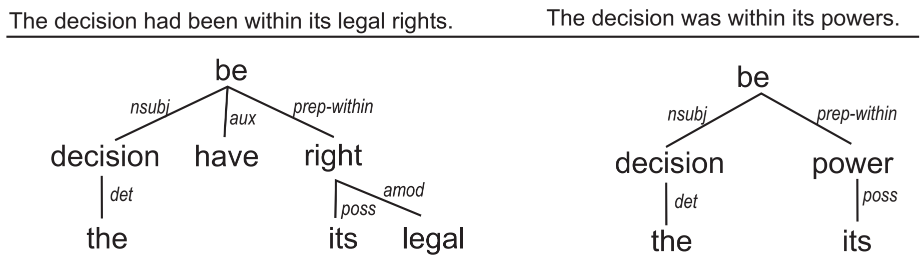
On the other hand, there are sentences that are almost identical, lexically and syntactically, and yet they are not paraphrases because the few dissimilarities make a big difference. In the example below, there is a relatively “small” difference between the two sentences. Only the subject of the sentences is different. However, due to the importance of the subject relation to the meaning of any sentence the high similarity between the sentences is sufficiently dominated by the ”small” dissimilarity to make the two sentences non-paraphrases.

Text A: *CBS is the leader in the 18 to 46 age group.*

Text B: *NBC is the leader in the 18 to 46 age group.*

Thus, it is important to assess both similarities and dissimilarities between two sentences S_1 and S_2 before making a decision with respect to them being paraphrases or not. In our approach, we capture the two aspects, similarity or dissimilarity, and then find the dominant aspect by computing a final paraphrase score as the ratio of the similarity and dissimilarity scores:

$$Paraphrase(S_1, S_2) = Sim(S_1, S_2)/Diss(S_1, S_2)$$



Paired Dependencies:
det(decision, the) = det(decision, the)
nsubj(be, decision) = nsubj(be, decision)
poss(power, its) = poss(right, its)
prep_within(be, power) = prep_within(be, right)

Unpaired Dependencies/Sentence 1:
aux(be, had)
amod(right-n, legal-a)
Unpaired Dependencies/Sentence 2:
EMPTY

Figure 5.1
Example of dependency trees and sets of paired and non-paired dependencies.

If the paraphrase score is above a learned threshold T the sentences are deemed paraphrases. Otherwise, they are non-paraphrases.

The similarity and dissimilarity scores are computed based on dependency relations (Hays 1964), which are asymmetric relationships between two words in a sentence, a *head* and a *modifier*. A sentence can be represented by a set of dependency relations (see the bottom half of Figure 5.1). An example of dependency is the *subject* relation between *John* and *drives* in the sentence *John drives a car.* Such a dependency can be viewed as the triple *subj(John, drive)*. In the triplets the words are lemmatized, i.e., all morphological variations of a word are mapped onto its base form. For instance, *go, went, gone, going* are all mapped onto *go*.

The Sim(S_1, S_2) and Diss(S_1,S_2) scores are computed in three phases: (1) map the input sentences into sets of dependencies, (2) detect common and non-common dependencies between the sentences, and (3) compute the Sim(S_1, S_2) and Diss(S_1,S_2) scores. Figure 5.2 depicts the general architecture of the system in which the three processing phases are shown as the three major modules.

In the first phase, the set of dependencies for the two sentences is extracted using a dependency parser. We use both Minipar (Lin 1993) and the Stanford parser (de Marneffe, MacCartney, and Manning 1993) to parse the sentences. Because these parsers do not produce perfect output the reader should regard our results as a lower bound, i.e., results in the presence of parsing errors. Should the parsing been perfect, we expect our results to look better. The parser takes as input the raw sentence and returns as output a dependency tree (Minipar) or a list of dependencies (Stanford). In a dependency tree, every word in the sentence is a modifier of exactly one word, its head, except the head word of the sentence, which does not have a head. The head word of the sentence is the root node in the dependency tree. Given a dependency tree, the list of dependencies can be easily derived by traversing the tree and for each internal node, which is head of at least one dependency, we retrieve triplets of the form *rel(head, modifier)* where *rel* represents the type of dependency that links the node, i.e., the *head*, to one of its children, the *modifier*. Figure 1 shows the set of dependencies in the form of triplets for the dependency trees in the top half of the figure.

In this phase, we also gather positional information about each dependency in the dependency tree as we will need this information later when weighting dependencies in Phase 3. The position/depth of a dependency within the dependency tree is calculated as the distance from the root of the node corresponding to the head word of the dependency. Because the Stanford parser does not provide the position of the dependencies within the tree, we had to recursively reconstruct the tree based on the given set of dependency relations and calculate the relative position of each relation from the root.

The second phase in our approach identifies the common and non-common dependencies of the sentences, based on word semantics and syntactic

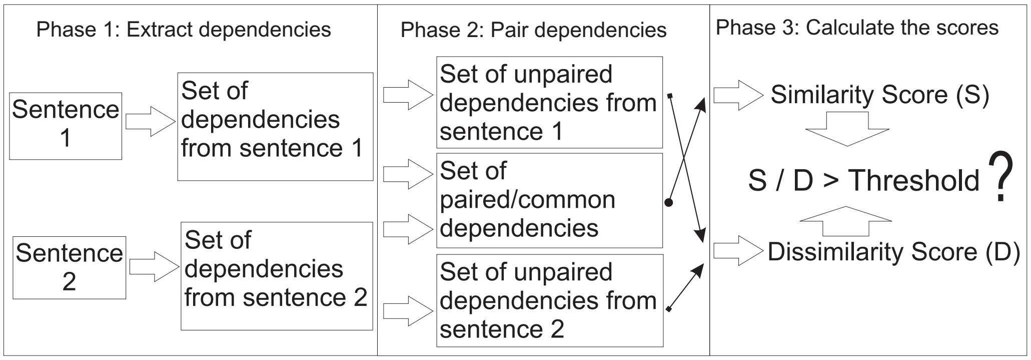

Phase 1: Extract dependencies
Sentence 1
Set of dependencies from sentence 1
Sentence 2
Set of dependencies from sentence 2
Phase 2: Pair dependencies
Set of unpaired dependencies from sentence 1
Set of paired/common dependencies
Set of unpaired dependencies from sentence 2
Phase 3: Calculate the scores
Similarity Score (S)
S / D > Threshold ?
Dissimilarity Score (D)

Figure 5.2
Architecture of the system.

information. Three sets of dependencies are generated in this phase: one set of *paired*/common dependencies and two sets of *unpaired* dependencies, one corresponding to each of the two sentences. To generate the paired and unpaired sets a two-step procedure is used. In the first step, we take one dependency from the shorter sentence in terms of number of dependencies (a computational efficiency trick) and identify dependencies of the same type in the other sentence. In the second step, we compute a dependency similarity score (d2dSim) using the word-to-word similarity metrics applied to the two heads and two modifiers of the matched dependencies. Heads and modifiers are mapped onto all the corresponding concepts in WordNet, one concept for each sense of the heads and modifiers. The similarity is computed among all senses/concepts of the two heads and modifiers, respectively, and then the maximum similarity is retained. If a word is not present in WordNet exact matching is used. The word-to-word similarity scores are combined into one final dependency-to-dependency similarity score by taking the weighted average of the similarities of the heads and modifiers. Intuitively, more weight should be given to the similarity score of heads and less to the similarity score of modifiers because heads are the more important words. Surprisingly, while trying to learn a good weighting scheme from the training data we found that the opposite should be applied: more weight should be given to modifiers (0.55) and less to heads (0.45). We believe this is true only for the MSR Paraphrase Corpus and this weighting scheme should not be generalized to other paraphrase corpora. The MSR corpus was built in such a way that favored highly similar sentences in terms of major content words (common or proper nouns) because the extraction of the sentences was based on keyword searching of major events from the web. With the major content words similar, the modifiers are the heavy lifters when it comes to distinguishing between paraphrase and non-paraphrase cases. Another possible approach to calculate the similarity score between dependencies is to rely only on the similarity of the most dissimilar items, either heads or modifiers. We also tried this alternative approach, but it gave us slightly poorer results (around 2% decrease in performance), and therefore, using a weighted scheme to calculate the similarity score for dependencies proved to be a better choice. The dependency-to-dependency similarity score needs to exceed a certain threshold for two matched dependencies to be deemed similar. Empirically, we found out from training data that a good value for this threshold would be *0.5*. Once a pair of dependencies is deemed similar, we place it into the paired dependencies set, along with the calculated dependency-to-dependency similarity value. All the dependencies that could not be paired are moved into the unpaired dependencies sets.

$$sim(S_1, S_2) = \sum_{d_1 \in S_1} max_{d_2 \in S_2^*}[d2dSim(d_1, d_2)]$$

$$diss(S_1, S_2) = \sum_{d_1 \in unpS_1} weight(d_1) + \sum_{d_2 \in unpS_2} weight(d_2)$$

In the third and final phase of our approach, two scores are calculated from the three dependency sets obtained in Phase 2: a cumulative *similarity score* and a cumulative *dissimilarity score*. The cumulative similarity score Sim(S_1, S_2) is computed from the set of paired dependencies by summing up the dependency-to-dependency similarity scores (S_2^* in the equation for similarity score represents the set of remaining unpaired dependencies in the second sentence). Similarly, the dissimilarity score Diss(S_1, S_2) is calculated from the two sets of unpaired dependencies. Each unpaired dependency is weighted based on two features: the depth of the dependency within the dependency tree and type of dependency. The depth is important because an unpaired dependency that is closer to the root of the dependency tree, e.g., the main verb/predicate of sentence, is more important to indicate a big difference between two sentences. In our approach, each unpaired dependency is initially given a perfect weight of 1.00, which is then gradually penalized with a constant value (*0.20* for the Minipar output and *0.18* for the Stanford output), the farther away it is from the root node. The penalty values were derived empirically from training data. Our tests show that this particular feature works well only when applied to the sets of unpaired dependencies. The second feature that we use to weight dependencies is the type of dependency. For example a *subj* dependency, which is the relation between the verb and its subject, is more important to decide paraphrasing than a *det* dependency, which is the relation between a noun and its determiner. Each dependency type is assigned an importance level between 0 (no importance) and

1 (maximum importance). The importance level for each dependency type has been established by the authors based on their linguistic knowledge and an analysis of the role of various dependency types in a subset of sentences from the training data.

Before comparing the similarity and dissimilarity scores, we consider one more feature that will affect the dissimilarity score. This improvement, of a more statistical nature, is based on the idea that if one sentence contains a significant amount of extra information compared to the other sentence although they do refer to the same action or event, then the relation between the two sentences is not a bidirectional relation of paraphrase, but rather a unidirectional relation of entailment, so they should be evaluated as non-paraphrases. This extra information is recorded in our dependency sets by the fact that the set of unpaired dependencies from the longer, more detailed sentence is larger than the set of unpaired dependencies from the shorter sentence. To account for this statistical feature, we add an absolute value to the dissimilarity score, which was empirically chosen to be *14*, for every case when the set of unpaired dependencies from the longer sentence has more than *6* extra dependencies compared to the set of unpaired dependencies from the shorter sentence. We chose these optimal constants values to tweak this feature, based on a series of tests made on the MSR Paraphrase Corpus, and because of that, by including it into the system, the performance was improved significantly.

Once the Sim(S_1, S_2) and Diss(S_1, S_2) scores are available, the paraphrase score is calculated by taking the ratio between the similarity score, S, and the dissimilarity score, D, and compare it to the optimum threshold T learned from training data. Formally, if $S/D > T$ then the instance is classified as paraphrase, otherwise is a non-paraphrase. To avoid division by zero for cases in which the two sentences are identical ($D = 0$) the actual implementation tests for $S > T * D$. To find the optimum threshold, we did an exhaustive search on the training data set, looking for the value which led to optimum accuracy. This is similar to the sigmoid function of the simple voted perceptron learning algorithm used in (Corley and Mihalcea 2005).

5.3. Summary of Results

We experimented with our approach on the MSR Paraphrase Corpus (Dolan, Quirk, and Brockett 2004), which was previously introduced in Section 1.6.1. To review, the MSR Paraphrase Corpus is the largest publicly available annotated paraphrase corpus which has been used in most of the recent studies that addressed the problem of paraphrase identification. The corpus consists of 5801 sentence pairs collected from newswire articles, 3900 of which were labeled as paraphrases by human annotators. The whole set is divided into a training subset (4076 sentences of which 2753 are true paraphrases) which we have used to determine the optimum threshold T, and a test subset (1725 pairs of which 1147 are true paraphrases) that is used to report the performance results. We report results using four performance metrics: accuracy (percentage of instances correctly predicted out of all instances), precision (percentage of predicted paraphrases that are indeed paraphrases), recall (percentage of true paraphrases that were predicted as such), and f-measure (harmonic mean of precision and recall).

In Table 5.1 three baselines are reported: a uniform baseline in which the majority class (paraphrase) in the training data is always chosen, a random baseline taken from (Corley and Mihalcea 2005), and a lexical baseline taken from (Zhang and Patrick 2005) which uses a supervised learning decision tree classifier

Table 5.1
Performance and comparison of different approaches on the MSR Paraphrase Corpus.

System	Acc.	Prec.	Recall	F-score
Uniform baseline	0.6649	0.6649	1.0000	0.7987
Random baseline (Corley&Mihalcea'05)	0.5130	0.6830	0.5000	0.5780
Lexical baseline (Zhang&Patrick'05)	0.7230	0.7880	0.7980	0.7930
Corley and Mihalcea (2005)	0.7150	0.7230	0.9250	0.8120
Qiu (2006)	0.7200	0.7250	0.9340	0.8160
Rus (2008a) - average	0.7061	0.7207	0.9111	0.8048
Simple dep. overlap (Minipar)	0.6939	0.7109	0.9093	0.7979
Simple dep. overlap (Stanford)	0.6823	0.7064	0.8936	0.7890
Optimum results (Minipar)	0.7206	0.7404	0.8928	0.8095
Optimum results (Stanford)	0.7101	0.7270	0.9032	0.8056
No word semantics (Minipar)	0.7038	0.7184	0.9119	0.8037
No word semantics (Stanford)	0.7032	0.7237	0.8954	0.8005
No dependency weighting (Minipar)	0.7177	0.7378	0.8928	0.8079
No dependency weighting (Stanford)	0.7067	0.7265	0.8963	0.8025
No penalty for extra info (Minipar)	0.7067	0.7275	0.8936	0.8020
No penalty for extra info (Stanford)	0.7032	0.7138	0.9241	0.8055

with various lexical-matching features. We next show the results of others including results obtained using the simple dependency overlap method in (Lintean, Rus, and Graesser 2008). The simple dependency overlap method computes the number of common dependency relations between the two sentences divided by the average number of relations in the two sentences. Our results are then presented in the following order: our best/state-of-the-art system, that uses all three features described in the previous section: word semantics, weighted dependencies and penalties for extra information, then a version of the proposed approach without word semantics (similarity in this case is 1 if words are identical, case insensitive, or 0 otherwise), then one without weighted dependencies, and finally, one version where the instances with extra information found in one of their sentences are not penalized.

The conclusion based on our best approach is that a mix of word semantics and weighted dependencies leads to better accuracy and in particular better precision. The best approach leads to significantly better results than the naive baselines and the simple dependency overlap ($p<0.001$ for the version with Minipar). The comparison between our best results and the results reported by (Corley and Mihalcea 2005) and (Lintean, Rus, and Graesser 2008) is of particular importance. These comparisons indicate that weighted dependencies and word semantics leads to better accuracy and precision than using only word semantics (Corley and Mihalcea 2005) or only simple dependency overlap (Lintean, Rus, and Graesser 2008).

All results in Table 5.1 were obtained with the *lin* measure from the WordNet similarity package, except the case that did not use WordNet similarity measures at all – the *No word semantics* row. This *lin* measure consistently led to the best performance in our experiments when compared to all the other measures offered by the WordNet similarity package.

For reference, we report in Table 5.2 results obtained when various word-to-word similarity metrics are used with an optimum threshold calculated from the *test data set*. For *lin* measure we report results with optimum test thresholds when using both parsers, Minipar and Stanford, while for the rest of the measures we only report results when using Minipar. We deem these results as one type of benchmark results for approaches that rely on WordNet similarity measures and dependencies as they were obtained by optimizing the approach on the testing data. As we can see from the table, the results are not much higher than the results in Table 5.1 where the threshold was derived from training data.

Table 5.2
Accuracy results for different WordNet metrics with optimum test threshold values

Metric	Acc.	Prec.	Rec.	F
$\text{Lin}_{Minipar}$	.7241	.7395	.9032	.8132
$\text{Lin}_{Stanford}$	.7130	.7387	.8797	.8030
Path	.7183	.7332	.9058	.8105
L & C	.7165	.7253	.9233	.8124
W & P	.7188	.7270	.9241	.8138
J & C	.7217	.7425	.8901	.8097
Lesk	.7148	.7446	.8692	.8021
Vector	.7200	.7330	.9093	.8117
Vector pairs	.7188	.7519	.8614	.8029

One important advantage that our system has over other approaches (Zhang and Patrick 2005; Qiu, Kan, and Chua 2006) is that it does not rely too much on the training. The training data is used merely to tune the parameters, rather than for training a whole classifier. Since the only parameter whose value fully depends on the training data is the final threshold value, we've made another set of experiments where the threshold value depends only on one piece of information about the characteristic of the test data set: the percentage of paraphrase instances within the data set. In other words, when calculating the threshold value, the system needs only know only what the probability of finding a paraphrase is within the given data set. The system then tries to find a threshold value that splits the instances into two sets with the same distribution of instances as the given data set. For the testing part of the MSR Paraphrase data corpus the distribution value is 0.6649. We used this information to decide on a threshold and the results were no more than 2.09% below the optimum performance scores (for example on Minipar output and when excluding the WordNet similarity feature, the accuracy performance was only 0.06 percent less than when the threshold is calculated from the training data).

5.4. Using IDF-based weighting on Dependency Relations

Another idea to improve the performance of our method was already suggested in section 3.9. The idea is to use the specificity of words (e.g., IDF) and weight them when calculating the similarity and dissimilarity scores. As already mentioned in previous chapter, we rely on the assumption that when a word is considered highly specific it should play an important role when deciding for how semantically similar two texts are. To further motivate this assumption, we show below a pair of sentences extracted from the MSR test data (instance #89), where by using IDF, our method successfully classify an otherwise failed instance:

> Text A: *Emily Church is London bureau chief of CBS.MarketWatch.com.*
> Text B: *Russ Britt is the Los Angeles Bureau Chief for CBS.MarketWatch.com.*

Notice that even though the predicates are the same and there is a rather long common noun phrase, which results in a significant number of identical dependencies between the two sentences, the subjects and the locations are completely different. Because there are two different pairs of names with high IDF values, this will put a significant weight on the dissimilarity score, which in the end will affect the decision that the two sentences are in fact not paraphrases.

We use the IDF values calculated from Wikipedia. The process for extracting these values is explained in detail earlier, in Section 3.9. We apply IDF-based weights on the paired dependencies when calculating the similarity scores, and similarly, IDF-based weights on the unpaired dependencies when calculating the

Table 5.3
Performance scores when using IDF values from Wikipedia.

Method	Parser	Optimum train threshold			Optimum test threshold		
		Acc.	Prec.	Recall	Acc.	Prec.	Recall
Sim&Diss	Minipar	0.6922	0.7133	0.8980	0.6957	0.7418	0.8317
	Stanford	0.7049	0.7228	0.9024	0.7101	0.7289	0.8980
Diss only	Minipar	0.7049	0.7450	0.8457	0.7113	0.7246	0.9128
	Stanford	0.7043	0.7323	0.8753	0.7072	0.7242	0.9041

dissimilarity scores. The weights for paired and unpaired dependencies, respectively, are calculated according with the following formulas:

$$W_{idf}(d_{(w_1,w_2)}, d_{(w_3,w_4)}) = [\sum_{i=1}^{4} idf(w_i)]/(4 * 6)$$

$$W_{idf}(d_{(head,mod)}) = [idf(head) + idf(mod)]/(2 * 6)$$

We experimented two approaches for using the IDF weights: 1) apply IDF weights to both paired and unpaired dependencies 2) apply IDF weights only to unpaired dependencies.

Table 5.3 shows results on these two methods when used with both dependency parsers (Minipar and Stanford). We show the same performance scores as in the previous section on optimum thresholds derived from both the training and the testing data sets. An interesting observation when looking at these results is that the first IDF method works better when used on the Minipar parser, while the second method works better on the Stanford parser. Another observation when comparing these results with Table 5.1 is that for the second IDF method, the precision values seem to increase, at the expense of lower recall.

5.5. Discussions

In addition to the previously raised issues, discussed in Section 1.6.1, regarding the annotation of the MSR corpus, another item worth discussing here is on the comparison of the dependency parsers. Our experimental results show that Minipar consistently outperforms Stanford, in terms of accuracy of our paraphrase identification approach. Minipar is also faster than Stanford[1], which first generates the phrase-based syntactic tree for a sentence and then extracts the corresponding sets of dependencies from the phrase-based syntactic tree. For instance, Minipar can parse 1725 pairs of sentences, i.e., 3450 sentences, in 48 seconds while Stanford parser takes 1926 seconds, i.e., 32 minutes and 6 seconds.

5.6. Importance of WordNet Similarity for Dependency Relations

Following from previous chapter we further motivate in here how WordNet similarity measures are useful when deciding on the presence of paraphrase relations.

> Text A: *They say second-quarter earnings reports will be key in giving investors that guidance.*
> Text B: *The upcoming second-quarter earnings season will be particularly important in offering investors guidance, they say.*

In the above example (instance number 103 from the testing part of the MSR corpus), there are several dependencies that are paired with the help of the WordNet similarity metrics. The similarity of words *key* and *important* or *offering* and *guidance* is decided using WordNet similarity metrics and thus the

[1] This was also mentioned before, in Section 2.6.

corresponding dependencies in which these words are involved are correctly paired, leading to more accurate overlap scores for the two sentences.

CHAPTER 6

KERNEL-BASED METHODS

The final set of methods, which are proposed in this dissertation for the task of semantic similarity assessment, are called *kernel-based methods*, because they rely on various predefined kernel functions, used in conjunction with support vector machine (SVM) classifiers, to qualitatively assess semantic relations between texts. One main difference between the methods presented in this chapter and those studied in previous chapters is that kernel-based methods rely on supervised based learning that takes full advantage of any lexical, semantic or syntactic features which may be seen in the training set and found to be relevant to the task of semantic similarity assessment. In our previous methods, the classification component relied only on the computed semantic similarity scores to learn threshold values, which classify the data instances in two groups, depending on whether a relation of semantic similarity is found or not.

We start this chapter by introducing the SVM classifiers and text-based kernels. Then we study some simple kernels based on the lexical similarities and differences between two input texts. In a sense, we will be extending some of the metrics presented in chapter 3. Then we will take a look at some more complex kernels based on dependency relations that we previously studied in chapter 5. Specifically we shall compare the trees of dependencies characterizing the input texts.

In this chapter we focus on text-to-text symmetric similarities only. Preliminary experiments have shown us that using these kernels-based methods on asymmetric relations, such as entailment, do not work well, even when we make them account for the difference in role between the source (i.e., the entailed text) and the target (i.e., the entailed hypothesis). Most of the evaluation work presented in this chapter was done on the Microsoft Research Paraphrase Corpus (see Section 1.6.1). We will also show some partial results made on ULPC, the other dataset on symmetric similarity relations that we introduced in the first chapter of this dissertation (see Section 1.6.2).

The following section presents the theory behind Support Vector Machines learning and further motivates why are kernel methods so popular in solving problems dealing with classification of natural language based inputs, especially on the topic of document classification. For even more details on SVM learning, we refer to the book of Vapnik on Statistical Learning Theory (Vapnik 1998) for a thoroughly description and mathematical analysis of SVMs, or the Burges' tutorial on SVMs (Burges 1998) for a quicker understanding of the SVM learning theory. The works of Zanzotto, Pannacchiotti, and Moschitti (2009) and Moschitti (2009) can also be consulted for further discussions on kernels used in paraphrase and textual entailment recognition tasks.

6.1. Support Vector Machines

Support Vector Machines (Cristianini and Shawe-Taylor 2000, SVMs) are one class of supervised, function-based machine learning methods that can be used for classification and regression tasks. In the simple case of binary classification, the intuition behind SVM learning is to project the training data into a multidimensional space, where there exist a hyperplane that can optimally divide the data points into two groups, each specific to one of the two classes, into which the data needs to be classified. SVM classifiers aim at finding a hyperplane that simultaneously minimizes the empirical classification error and maximizes the

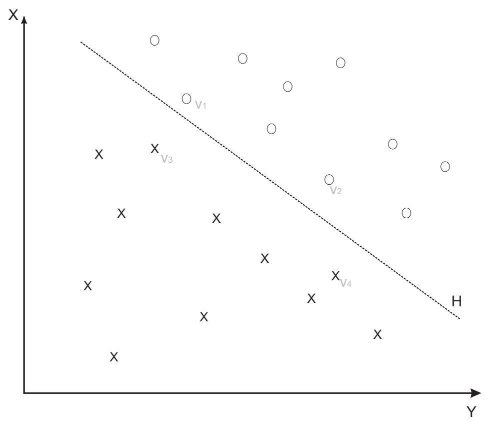


Figure 6.1
Two-class separation hyperplane in a bidimensional space

geometric margin between the hyperplane and the nearest data points. Maximum-margin hyperplanes lead in general to smaller generalization errors, a desirable outcome. *Support vectors* are data points which affect how the hyperplanes are chosen. In the fortunate case, when the data is linearly separable (see Figure 6.1 for data separation in a bidimensional space) the hyperplane is chosen in such a way that the support vectors (i.e., v_1, v_2, v_3, v_4) are equally distanced from the plane on either side. This type of classification, through the use of separable hyperplanes, gives one very useful feature to the SVMs, and that is the ability to handle outliers really well, since outliers represent data points which are located far away from the separation hyper plane, hence they will not count as support vectors that decide how the hyperplane is calculated.

One simple way to project our data into a multi-dimensional space is to assume one dimension for each numeric feature characterizing the data instances. If there are 10 such features, then there will be 10 dimensions for the projected space, which is called the *feature space*. To search for a separating hyperplane in the feature space, the SVM classifier employs the use of a linear kernel function, computed between any two data points, which essentially reflects how close two instance points are in that space. The linear kernel is defined as the inner product between the vectors representing the data points - $K_{linear}(x, y) = (x \cdot y)$. In most machine learning problems however, the data is not linearly separable in the future space. In such cases, the data has to be projected into a new multi-dimensional space where, hopefully, it will be linearly separable. As is turns out in SVM learning, knowing the relative distance between the data points (i.e., the output of the kernel function) is enough to define the space and find the separating hyperplane which is being represented by its support vector points. Therefore, SVMs do not require any form of representing the data instances in this multidimensional space where the learning occurs. Furthermore, changing this space can be done by simply employing a different kernel function as it is described in the following subsection.

6.1.1 Kernel Functions for SVMs

To move from linear separation to non-linear separation, various kernel functions can be defined to project the initial feature space into a projected space, which is usually comprised of a higher number of dimensions, where a linear separation can be found. Some classic examples of such kernels are: the polynomial kernel - $K_{poly}(x, y) = (x \cdot y + coef)^d$, radial basis function - $K_{rad}(x, y) = exp(-\gamma||x - y||)^2$, or two layer sigmoid function - $K_{sig}(x, y) = tanh(\gamma xy + coef)$ (which makes the SVM similar to a perceptron classifier). A kernel function can help solve classification problems that might require complex separation hyperplanes. That is, it can map problems for which a polynomial or more complex separation hyperplane may be needed into linear classification problem. This is possible because kernel functions map the input space into a new, highly-dimensional feature space, where linear separation may be possible. The hardest part is finding a good kernel that is also valid for SVM learning.

A valid kernel function $K(x, y)$ has to respect Mercer's conditions, which sais that for all square integrable functions $g(x)$, the following inequality holds:

$$\int\int K(x, y)g(x)g(y)dxdy \geq 0 \tag{6.1}$$

A particular case of the above condition stipulates that a kernel function that is symmetric continuous and non-negative definite, respects Mercer's theorem. That is, the following inequality is true for all finite sequences of points $x_1, ..., X_n$, and all choices of real numbers $c_1, ..., c_n$:

$$\sum_{i=1}^{n} sum_{j=1}^{n} K(x_i, x_j)c_i c_j \geq 0 \tag{6.2}$$

It can be easily proven mathematically that functions, which are directly derived from the inner product of vectors in an Euclidian space (or Hilbert space, to be more general), are in fact non-negative definite (or positive semidefinite). As we shall see, most kernels that are being proposed in this work, can be easily associated to inner products in some predefined multi-dimensional space. As Lodhi pointed out (Lodhi et al. 2002), for these kernels there is no need to further prove that they satisfy Mercer's conditions (i.e., they are symmetric and non-negative defined), since they follow automatically from the definition of the inner product.

An intuitive way to understand how these nonlinear kernels work is by considering the simple case of binary classification of data points on a surface. Let us consider any 3 points, distinctly located on this surface, that are not collinear (i.e., they are not located on the same line). Then, any labeling of these three points can be correctly separated by a line (which is actually a hyperplane in the bidimensional space). When a fourth point is added with an arbitrary labeling, then a separation with only one line will not be possible in all cases. However, when we project these four points into a three dimensional space, then again we can find a plane that can correctly separate the points, base on the class labels that was assigned to them. A polynomial kernel of second degree - $K(x, y) = (x \cdot y)^2$, can easily project the data points from a surface into a three dimensional space.

The space in which a kernel projects the initial data is usually hard to process directly, due to the very high number of dimensions (i.e., some kernels can project the initial data into a space with an infinite number of dimensions. Therefore SVMs rely directly on the kernel functions during both training and classification phase. This requires the ability to solve a quadratic programming optimization problem with linear constraints and techniques and programs for solving it already exists. The trick is to be able to efficiently compute the kernel function.

6.1.2 The VC dimensions of SVMs: overfitting vs. generalization

A major advantage when using SVMs in machine learning (ML), is that we can decide on a tradeoff between the machine's *capacity* to learn and assimilate most of the information given in the training data, and its ability to generalize efficiently, on new unseen data. This tradeoff is made possible because of the relative freedom that we have, in choosing the number of dimensions for the space in which data will be projected. These dimensions are derived from the set of features which define the data. If the number of dimensions is set too high, then the machine can learn on the given data very well but might perform worse on new instances (i.e., there is a risk of overfitting the data). In case of data derived from natural language input, the number of features on the data instances representing the input texts can be very high. Consider for example the set of all the words, concepts or situations which might be contained in a text instance. The goal in this case is to find a representation that is able to assimilate all these different concepts and situations, but still be able to generalize well for new instances. In other words, a good balance is desirable between allowing a high capacity to assimilate many different situations, but at the same time not overfit the data too much.

The second feature that SVMs offer is the ability to define a bound with a certain probability, on the error rate for new data. Consider the simple case of binary classification. Based on an *empirical risk* (defined as R_{emp} in equation 6.3), which represents the calculated error for the training set, Vapnik (Vapnik 1998) proved that a bound, called the *risk bound* (defined as R in equation 6.4) can be computed for the estimated error on the test set. The risk bound holds with a chosen probability of $1 - \mu$ and depends on the *Vapnik Chervonenkis (VC) dimension* (defined as the non-negative integer h in equation 6.5). The VC dimension characterizes the capacity of the machine, which we discussed in previous paragraph. It indicates what is the maximum number of training points, Max_θ which can be *shattered* by a class of functions, θ (i.e., we can always find a set of Max_θ points, for which, no matter what combination of class values is being assigned to them, there exists a function in θ, that can cover them).

$$R_{emp}(\alpha) = \frac{1}{2l}\sum_{i=1}^{l}|y_i - f(x_i, \alpha)| \tag{6.3}$$

$$R(\alpha) = \int \frac{1}{2}|y - f(x, \alpha)|dP(x, y) \tag{6.4}$$

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\frac{h(log(2l/h)+1) - log(\mu/4)}{l}} \tag{6.5}$$

One important aspect regarding the risk bound, which needs further commenting, is that it predicts the error in worst case scenario, but does not guarantee it. From equation 6.5 it is easy to observe that, when the chosen probability of $1 - \mu$ for the error bound is set to 1 (meaning 100% assurance that the bound will hold), the bound actually goes to infinity (i.e., $-log(\mu/4) = -log0 = \infty$). However, if we are willing to take a calculated risk (say for example, $1 - \mu = .95$), then we are assured, with a certain degree of chance, that on unseen data the SVM will not fail more than the estimated error bound.

Being able to calculate the risk bound offers a good starting point when choosing between different families of kernel functions and is also the main idea behind the theory of *structural risk minimization* (Shawe-Taylor et al. 1996). Consider having the option to choose from several classes of functions, for which we can separately compute their corresponding VC values. The idea is to choose one function from each group (and a VC) and calculate the empirical risk from which we can then approximate the risk bound. By comparing the approximated risk bounds, one can gain useful insight on which group of functions is most appropriate for the classification. This approach is recommended as opposed to the other classic options that empirically decide, throughout experimentation, which class of functions is best, for a particular learning task.

To get a better intuition on the real meaning of the VC value that is characterizing a given kernel function, we must take a look at the space in which the function is projecting the input data. For example, let us take the previous simple case of a bidimensional space, where the separation hyperplane is defined by a line, the VC value of any function that is projecting the data into this space, is 3. This means we can find a maximum of 3 points in this space, which can be *shattered* by these functions, but not 4 or more. In general, in an n-dimensional space, the VC would be n+1 (Anthony and Biggs 1995). Based on the formula for calculating the risk bound, if we want to minimize the bound, then we need to set the VC to a value that is as low as possible, which in consequence requires that the number of dimensions to be low too. As long as the accuracy on the training set is acceptable, the lower the number of dimensions defined, the tighter the bound, suggesting that there is a lower risk of overfitting the data when there are less dimensions. This is also in accordance with the Occam’s razor’s law which tells us that the simplest explanation is often the best one.

6.1.3 SVMs for NLP related tasks

Due to their ability to work efficiently in highly-dimensional spaces, SVMs are a good fit for tackling natural language processing problems such as the ones addressed in this dissertation. In general, data that is derived from textual inputs is characterized by a high number of features, which are extracted from the lexical tokens composing the input texts and the syntactic relations between them. In addition, SVMs ability to efficiently handle outliers and noise also goes well with the nature of most NLP related tasks where, whenever some kind of natural language based input is expected, the presence of irrelevant noise or ”special” input instance, which do not conform to the rest of the data, is very probable.

SVMs can be used in two ways for learning and classification tasks. One is similar to other machine learning approaches (e.g., Bayesian classifiers, decision trees), by using standard kernels (like the ones previously listed in subsection 6.1.1) on a set of numeric features extracted from the data. We call this the *classic*

SVM approach. The other way is by developing customized kernels which are more appropriate and specific to data of some particular learning task. For example, most kernels that are used in NLP related tasks make use of vocabularies of words to explode the number of dimensions for the space in which the textual input data is projected. We call this the *kernel-based approach*. For the first approach, the challenge is to find a good set of features to represent the data and an optimum set of parameters for the learning algorithm. The second approach requires finding a good and valid kernel function that is fitted for the data and that can be computed in a fast and efficient manner, in order to make the learning process practically feasible.

In previous work related to the task of measuring semantic similarity between texts, we see that SVMs are commonly used with the classic approach. Dolan and Brockett (Dolan, Quirk, and Brockett 2004; Brockett and Dolan 2005) used SVMs in the creation of the MSR, the main corpus which we frequently use in our current work[1]. Furthermore people have successfully used SVMs on the same corpus, for the problem of paraphrase identification (Qiu, Kan, and Chua 2006; Wan et al. 2006; Das and Smith 2009).

The real focus of the work presented in this chapter regards the second approach, based on customized kernel functions. For NLP problems, kernel-based methods have been mostly used before on tasks of document clustering (Lodhi et al. 2002). In these tasks the input set of documents are represented as vectors in a high-dimensional space, defined by a kernel function, which measures the similarity between two documents, in regard to a relevant

[1] The other two corpora that we use in our experiments, the ULPC and the RTE, were, in most part, manually created and labeled by humans and human experts

classification factor (e.g., same topic of discussion, or same writing style). In this projected space, documents of a similar type should aggregate together in bulks of data that is linearly separable from data characterizing other document types.

We are particularly interested in a special type of text-based kernel functions, ranging from substring kernels, which compute a weighted-sum of common substrings in two documents in order to classify them (Lodhi et al. 2002), to dependency kernels, which compute dependency substring overlap (Kate 2008). Along the same lines, we define our own kernel that quantifies the degree of similarity and dissimilarity among two sentences given with each instance of our datasets. To the best of our knowledge, no one has approached the paraphrase identification task with a newly defined kernel function although standard kernels have been used, as we previously mentioned. For instance, Wan et al. (Wan et al. 2006) have used a small set of expert-defined features for the input space together with a polynomial kernel available in WEKA, a machine learning toolkit.

6.2. Kernels for Semantic Similarity Assessment

In this section we are going to propose various kernels which, in combination with SVM classifiers, will allow us to qualitatively assess symmetric relations of semantic similarity between texts. We are going to study two major groups of kernels. First, we look at some simple lexical kernels, based on counting common lexical tokens that are present in both input texts. Then we are going to look at common syntactic information found between the texts, which is encoded in their corresponding dependency trees.

In order to understand how these kernels work, we must compare them with other text-based kernels, in particular those used for text classification. In a classic problem of text classification (or clustering), the input is a text of medium or large size (e.g., news, emails, documents) and the output is a labeling of this text, according to some defined taxonomy of text-based concepts or categories. From a machine learning point of view, a data instance is represented by one block of input text (e.g., a paragraph or a document). A kernel function computes the similarity between two data instances, by counting the lexical, syntactic or semantic similarities found between their corresponding input texts. Based on this, same label instances that share many similarities will be located closer in the space projected by the kernel, which allows for a more probable linear separation of the data. One simple SVMs kernel for this type of problem is to count the number of common words or word sequences of two given instances. Such kernels are a generalization from string to word sequences of the string kernel proposed by Lodhi et al. (2002).

For our task, an instance of the input is represented by two pairs of text, instead of one. Therefore, a kernel function in this case computes the similarity between two pairs of texts, meaning that there are a total of four texts given as input to the kernel, as opposed our previous kernel, which had only two input texts. To compute similarities between two data instances (i.e., between two pairs of texts), we must first compare the texts from each instance, to extract relevant facts about the relation between them. Based on the relevant information extracted from the comparison of the texts, our kernel function will then use this to compare two instances of pairs, by looking at the similarities that the two instances share, similarly as for the kernels used in text classification tasks. The main assumption in this approach is that instances with many common similarities or common dissimilarities between the texts which they enclose, should be part of the same category or label class, which will confirm, or not, on the existence of a semantic relationship between the texts. In a sense, our SVMs are learning which of the information facts extracted from comparing the input texts is relevant for the similarity relation that we are trying to assess.

Next, we are going to detail on the two groups of kernels that we study in this chapter, for the task of measuring semantic similarity.

6.2.1 Lexical Kernels of Similarity and Dissimilarity

For our first group of kernels, we propose an approach to assess the similarity of lexical similarities or dissimilarities between two input texts. Qui, Kan and Chua (2006) previously suggested that in order to detect whether two input sentences are paraphrases, one should look instead at the differences between them and check if they are significant enough, for the sentences to be deemed semantically different. Let us look for example at the following pair of sentences, where the lexical differences are marked in italic font (we are comparing between only the base form of words):

A. Mary went to the doctor *yesterday.*

B. *I saw* Mary going to the doctor *the other day.*

These sentences can be considered to be paraphrases, since they state the same principal fact that Mary went to the doctor. The differences in this case merely present some additional details which may be discarded, in certain contexts. We expect that our SVMs should learn from examples like this, so that when they encounter similar differences on new data, they should not consider these differences as being the main cause for excluding the presence of a paraphrase relation in the data. Let us consider now the next pair of paraphrase sentences:

A. Josh bought some shoes from the mall.

B. *I saw* Josh buying some shoes at the mall *the other day.*

The system should recognize the differences from the previous example and ignore them when assessing the semantic similarity between the sentences. There is another difference in these sentences, between the two prepositions *from* and *at*, but these are also not significant enough for a difference in the semantic meaning of the two texts. A kernel function that allows an SVM classifier to learn about these types of differences, we call it a *lexical dissimilarity kernel*.

To implement such a lexical kernel based on dissimilarities, we first represent each instance (two sentences which may be or not in a paraphrase relation) in the dataset as a vector of dissimilarities. There is one dimension in the vector for each word or sequence of words in the input sentences. More accurately, the dimensions correspond to word-types, or sequences of word-types. Word-types are unique words in the input texts. The value along each dimension in this vectorial representation is non-zero if the corresponding word-type occurs in sentences that form an instance. If a word type or sequence is not present in an instance, the value is zero. Non-zero values can represent weights that measure the importance of the dimension/word-type for a particular instance. Similar to our previous methods, we can consider here for various weighting schemes, of either local type, global type, or both. Based on this representation, we define a kernel that efficiently computes the similarities between these vectors. That is, two instances that have similar dissimilarities should be projected close to each other in the feature space generated by this kernel.

We define the dissimilarity kernel value between two instances A and B of paired sentences (S_{A1}, S_{A2}) and (S_{B1}, S_{B2}) as in Equation 6.6. We represent sentences by their corresponding set of lexical tokens or word types (punctuation included). We denote with $S_{A1} \Delta S_{A2}$ the symmetric difference between the two sets, meaning the set of tokens that are present in S_{A1} or S_{A2} but not both, while $w_1 \equiv w_2$ means the tokens are found to be equivalent.

$$DisK(A, B) = \sum_{\substack{w_1 \in S_{A1} \Delta S_{A2} \\ w_2 \in S_{B1} \Delta S_{B2} \\ w_1 \equiv w_2}} weight(w_1) * weight(w_2) \tag{6.6}$$

The equivalence between the tokens can be judged from various angles, depending on what is compared. We can compare the initial forms of the tokens, or their base form or their part-of-speech. We could also define a relation of equivalence based on the W2W similarity metrics that were discussed in Chapter 4. In this case, if the computed similarity between two tokens is above a preset threshold value, then the tokens are considered to be equivalent, and not if otherwise. In this current study, we only consider the simplest relations of equivalence (i.e., compare words, lemmas, and parts-of-speech), leaving other more complex relations for future work.

The $weight$ function retrieves the weight for the corresponding token. In a previous study done on these types of lexical kernels we found that if we do not employ any type of local weighting scheme, such as frequency or log-type frequency (see section 3.4), gave best performance scores on the MSR dataset. As a consequence, we decided that we will not use any type of local weighting in this study, in order to keep our experiments simple, allowing us to focus on other parameters of the learning process, which might prove to be more important for our task. In future work, we might want to experiment with global weighting schemes also, such as entropy or inverse document frequency (Dumais 1991).

Consequently, we can define a *lexical similarity kernel* which is based on looking at the common lexical tokens that are found between two input texts, as in Equation 6.7.

$$SimK(A,B) = \sum_{\substack{w_1 \in S_{A1} \cup S_{A2} \\ w_2 \in S_{B1} \cup S_{B2} \\ w_1 \equiv w_2}} weight(w_1) * weight(w_2) \tag{6.7}$$

There is one very important aspect about how we computed these kernels, which is not apparent in our formulas. All the token that we consider are counted only once during the computation process, meaning that, in the sums referred in our formulas, each particular token in one instance is only used once, even if it is equivalent with more than one token from the other instance.

As we will see in our experiments the similarity kernel is not very powerful, since the important differences between the texts are ignored, and it will actually perform better when used in combination with the dissimilarity kernel, as indicated in Equation 6.8. We call this the kernel of similarities and dissimilarities, or in short, the *sim-diss kernel*.

$$SimDisK(A,B) = SimK(A,B) + DisK(A,B) \tag{6.8}$$

It can be easily proven that all these kernels are valid kernels for SVM learning (i.e., they conform to Mercer's condition), since there are inner products in the space where the input data is projected. As mentioned before, the dissimilarity kernel projects the input data in a space where the dimensions are represented by all n-gram types that are found different between two input texts, while the similarity kernel projects the data in a space where the dimensions are representations of all n-gram types that are found common between the texts.

Since any sum of two valid kernels is also recognized as a valid kernel, it can be concluded that the latter combination of previous two kernels is too a valid kernel.

A big advantage of the lexical kernels that were presented here is their time complexity, which is linear in the length of the input texts and ease of interpretation by humans as the dimensions correspond to words or word sequences.

6.2.2 Dependency-based kernels

The next group of kernels is more complex and is based on the overall syntactic organization of a sentence, which is encoded in its associated tree of dependency relations. The developing of these kernels were greatly inspired from the previous work of Kate and Mooney (2008), where a dependency-based word subsequence kernel was proposed to compute similarity between two given sentences as the number of paths shared by their dependency trees. The main difference in our kernels is that we compute similarity between two given pairs of inputs (so that is a total of four sentences).

Given a pair of two input sentences, A_1 and A_2, in a first step, we mark in each of their associated dependency trees, those nodes that are found to be common in both trees. Then we create a list, LP_{sim}, of all downward paths, starting from the root, that are found common between the two trees. Similarly we create a list, LP_{dis}, of all paths that are present in one tree, but not the other. Since we worked with symmetric similarity relations only, we include in this second list the distinct paths from both trees.

After creating the two lists of dependency paths we are going to use them in a kernel, just as we used the lists of common and distinct lexical tokens in our previous lexical-based kernels. Given two instances, A and B, composed each of two input sentences, A_1 and A_2, and respectively, B_1 and B_2, we are going to count, for a similarity dependency-based kernel, all paths that are found common in $LP_{sim}(A)$ and $LP_{sim}(B)$. Similarly, for a dissimilarity based, we are going to count all paths that are found common in $LP_{diss}(A)$ and $LP_{diss}(B)$.

Since both lists, LP_{sim} and LP_{dis}, can be preemptively constructed before starting the SVM learning process, the complexity of these kernels becomes linear in the total number of common and distrinct dependency paths found in all four input sentences. Likewise the case of lexical kernels, the dependecy kernels also express an inner product between vectors in a space where the dimensions are represented by the set of all common and distinct dependency-based paths that are found in the instances of a dataset.

6.2.3 Experiments and Results

We run a suite of experiments with the two types of lexical and dependency kernels and present how they perform on the MSR corpus. There are many options to set up these kernels. First, there are the various preprocessing options that we can select, some of which were previously presented in chapters 2 and 3. We decided to use the OpenNLP parser for the lexical kernels, and the Stanford parser for the dependency-kernels. In addition, we will use all of the extracted lexical tokens (punctuation included) when computing the lexical kernels.

Because there is a very high number of dimensions for the space where the data is projected by our kernels, we noticed that the SVMs perform very well on the training set, but they are less accurate on the testing. This suggests that the kernels have a tendency to overfit too much on the training data. In order to improve on the generalization of the SVMs classifiers we modified the *capacity* (C) factor, which is used by SVMs during the learning process (we previously introduced this factor in subsection 6.1.2 of the current chapter). To increase performance on the test set, we must give up some of the performance on the training set, and this can be done by lowering the C factor from its initial value of 1. We therefore experimented with three representative values of C: 1, 0.1 and 0.01. We found out that for values lower than these, the performance decreases on both sets of training and testing. We shall see in the reported results that for the most part, when the C-factor is lowered, the performance in the training decreases while the performance on the testing increases.

We report results on multiple variants for the lexical kernels: unigrams versus bigrams, comparing words versus lemmas versus parts-of-speech, and finally similarity versus dissimilarity versus a combination of both type of kernels. In all our experiments that were made for this chapter we use the latest version of LibSVM (Chang and Lin 2011), a java library for support vector machines. In a previous work (Lintean and Rus 2011), we reported similar results when evaluating the lexical kernels on the MSR dataset. In that work we experimented with a different SVM library, called SVM-Light (Joachims 1999), which is a much faster implementation of SVM classifier induction algorithm done in C. The reason why we did not use SVMLight for our current experiments was because the rest of our system was already done in Java, and LibSVM looked like a better option to incorporate into our system. We therefore tested our kernels with the LibSVM library and all the default learning parameters (e.g., the epsilon tolerance of termination criterion was left at its $.001$ default value) except the capacity factor, *C-factor*, mentioned in the previous paragraph.

The format in which we report the performance results for our kernels follows the format which we previously used in sections 3.8, 4.6, and 4.7, with one

Table 6.1
Lexical Kernels on MSR, with OpenNLP parsing and raw lexical forms

			Performance on Train			Performance on Test		
Kernel Type	C-factor	nSV	Acc.	Prec.	Recall	Acc.	Prec.	Recall
Sim	1.00	2864	.9814	.9880	.9844	.6649	.7542	.7358
	0.10	3086	.8400	.8179	.9818	.6962	.7086	.9224
	0.01	3065	.6904	.6876	.9924	.6806	.6776	.9913
Diss	1.00	2730	.9956	.9942	.9993	.7084	.7469	.8492
	0.10	2859	.8781	.8558	.9855	.7171	.7277	.9180
	0.01	2862	.6852	.6822	.9996	.6771	.6731	1.000
Sim-Diss	1.00	2964	1.000	1.000	1.000	.7136	.7631	.8256
	0.10	2997	.9652	.9611	.9884	.7345	.7612	.8753
	0.01	2933	.7107	.7020	.9935	.6881	.6829	.9913

difference. Instead of the classification threshold (which is not applicable for these kernel methods) we will show the number of support vectors learned by the SVM classifiers (nSV), since this will give us a good indication of how much overtraining is done and how slow will the classifier perform (since for every new case of classification, the new data instance is compared against all the support vectors which were learned before).

Table 6.1 report results on the MSR corpus, when using a lexical kernel, where the lexical tokens are compared using their raw, unchanged form. Three types of kernels are compared, as was previously mentioned: similarity versus dissimilarity versus the Sim-Diss kernel.

Our next tables, 6.2 and 6.3 report results on MSR, for the same lexical kernels, when the base form of words is used for comparison, and consequently, the part-of-speech..

From all these tables we can clearly see that the Sim-Diss kernel outperforms the simple ones. The most important result that we obtained from these experiments is that we managed to get perfect performance scores on the training

Table 6.2
Lexical Kernels on MSR, with OpenNLP parsing and base lexical forms

			Performance on Train			Performance on Test		
Kernel Type	C-factor	nSV	Acc.	Prec.	Recall	Acc.	Prec.	Recall
Sim	1.00	2814	.9715	.9779	.9800	.6539	.7455	.7280
	0.10	3005	.8187	.7988	.9778	.6945	.7053	.9285
	0.01	2999	.6916	.6886	.9920	.6806	.6776	.9913
Diss	1.00	2601	.9909	.9892	.9975	.6945	.7414	.8300
	0.10	2817	.8697	.8500	.9800	**.7188**	.7283	.9207
	0.01	2846	.6850	.6820	.9996	.6771	.6731	**1.000**
Sim-Diss	1.00	2874	**1.000**	1.000	1.000	.6957	.7504	.8126
	0.10	2909	.9531	.9470	.9858	.7183	**.7521**	.8596
	0.01	2886	.7130	.7037	.9931	.6899	.6846	.9895

Table 6.3
Lexical Kernels on MSR, with OpenNLP parsing and part-of-speech forms

			Performance on Train			Performance on Test		
Kernel Type	C-factor	nSV	Acc.	Prec.	Recall	Acc.	Prec.	Recall
Sim	1.00	2897	.6757	.6756	1.0000	.6649	.6649	1.0000
	0.10	2716	.6754	.6754	1.0000	.6649	.6649	1.0000
	0.01	2673	.6754	.6754	1.0000	.6649	.6649	1.0000
Diss	1.00	2532	.7188	.7382	.9045	.7188	.7348	.9032
	0.10	2555	.7179	.7337	.9139	.7194	.7320	.9119
	0.01	2662	.7002	.6954	.9895	.6829	.6820	.9799
Sim-Diss	1.00	2405	.7451	.7703	.8870	.7432	.7647	.8867
	0.10	2440	.7414	.7659	.8888	.7391	.7583	.8919
	0.01	2604	.7343	.7406	.9335	.7194	.7263	.9276

Table 6.4
Lexical Kernels on ULPC, with OpenNLP parsing

			Performance on Train			Performance on Test		
Kernel Type	C-factor	nSV	Acc.	Prec.	Recall	Acc.	Prec.	Recall
Raw Form	1.00	917	.8846	.8681	.9254	.7034	.7224	.7436
	0.10	1074	.7945	.7678	.8843	.7054	.6969	.8168
	0.01	1291	.7051	.6673	.8980	.6854	.6585	.8828
Base Form	1.00	915	.8606	.8400	.9142	.6974	.7133	.7473
	0.10	1058	.7812	.7511	.8856	.6914	.6854	.8059
	0.01	1297	.6991	.6606	.9030	.6754	.6496	.8828
Part-of-speech	1.00	1070	.6985	.6860	.8072	.6754	.6718	.7949
	0.10	1136	.6805	.6643	.8172	.6553	.6507	.7985
	0.01	1278	.6518	.6169	.9254	.6653	.6293	.9451

set, and still have decent performance scores on the test set (see our results when using Sim-Diss kernels with a C-factor of 1, in Tables 6.1 and 6.2) with an accuracy of $.7136$ on the best case, which is comparable to result obtained by others (Corley and Mihalcea 2005) on the MSR. Also we notice some very decent results obtained on both the training and testing sets by using a Sim-Diss kernel (C-factor = 1) with matching by part-of-speech ($.7451$ and $.7432$ on training and testing respectively). This clearly shows that these lexical kernels are very efficient in their learning of the various conditions that might be encoded in textual inputs.

Next, we tested the other corpus of symmetric similarity, the ULPC (see Table 6.4) but only with the Sim-Diss kernel, and on all three conditions of using raw, base and part-of-speech form, when comparing the tokens.

From the results on the ULPC we see that the kernels perform very well on both training and testing, when compared with our previous methods presented in chapters 3 and 4. On the testing part we obtained best accuracy ($.7054$), when matching the raw lexical forms with a C-factor of $.1$, best precision ($.7224$) on raw

Table 6.5
Dependency Kernels on MSR, with Stanford parsing

			Performance on Train			Performance on Test		
Kernel Type	C-factor	nSV	Acc.	Prec.	Recall	Acc.	Prec.	Recall
Diss	1.00	2695	.9630	.9543	.9927	.6180	.7017	.7402
	0.10	3045	.8994	.8768	.9902	.6597	.7044	.8413
	0.01	3137	.7571	.7391	.9898	.6852	.6876	.9651
Sim-Diss	1.00	2604	.9706	.9647	.9927	.6174	.7065	.7262
	0.10	2993	.9227	.9039	.9909	.6435	.7015	.8073
	0.01	3151	7951	.7721	.9884	.6823	.6924	.9398
Diss-base	1.00	3888	.9998	.9996	1.0000	.6945	.7037	.9337
	0.10	3934	.9971	.9964	.9993	.6980	.6996	.9564
	0.01	3973	.8565	.8267	.9964	.6870	.6834	.9861
Sim-Diss-base	1.00	3910	.9998	.9996	1.0000	.6887	.6981	.9372
	0.10	3925	.9988	.9989	.9993	.6916	.6988	.9425
	0.01	3964	.9264	.9039	.9971	.6922	.6883	.9817
Sim-Diss-POS	1.00	2944	.9983	.9982	.9993	.6458	.7233	.7568
	0.10	3289	.9831	.9772	.9982	.6678	.7271	.8012
	0.01	3420	.8550	.8260	.9949	.6916	.6943	.9582

forms with C-factor equal to 1, and best recall ($.9451$) when using part-of-speech tags with a C-factor of 0.01.

Regarding the dependency kernels, we tested these on MSR only (see Table 6.5, for comparison purposes, since we found out that they do not perform so well on the symmetric relations of semantic similarity, at least in the form that were currently defined. We looked at cases when comparing paths of only dependency types, for two variants of kernels, the dissimilarity and the Sim-Diss kernels (the similarity kernel was omitted, since we saw in previous experiments that it does not perform so well on its own). Then we enhanced the paths with information about the lexical forms that are closed by the dependency relations. Dissimilarity and Sim-Diss kernels were tested when using the base form of the tokens, and a Sim-Diss kernel was evaluated when using the part-of-speech tags of tokens.

Although, in theory, these types of kernels should be more powerful than the more simple ones, in practice they do not confirm this. We realize that there is a need for more research to be done on these types of kernels, which could also prove useful for cases when we wish to compute relations of asymmetric semantic similarity, such as entailment.

CHAPTER 7

CONCLUSIONS

Measuring semantic similarity between texts defines one very important class of problems in the field of natural language processing. The applications of such measures of text-to-text semantic similarity are abounding, from automatic understanding of natural language input, to information seeking and textual data mining and clustering. Plenty of research has been done so far on this topic and many methods have been proposed and researched. The current dissertation is an attempt to merge a lot of these ideas, and propose some new ones along, into a unique and well defined framework for representing the semantic meaning of the textual input and then exploit this representation to construct several methods for computing semantic similarity between texts.

In this dissertation we tried to give a broad overview of all the possibilities that one needs to be aware of when working with natural language input. Throughout our presented experiments we have shown readers a complete picture of the problem, with many detailed explanations, tables and numbers, which illustrate what it really means to fully evaluate a proposed method, given all the possible angles of a problem. Our experiments have shown how important is a simple step of preprocessing the input and how changing a small parameter in the method of preprocessing the data can mean the crucial difference in performance between a good and a mediocre system.

7.1. Previous Work

There has been a renaissance recently for exploring computational approaches to detect text-to-text semantic relations. The recent developments were driven primarily by the creation of standardized data sets for the major relations of entailment (Dagan, Glickman, and Magnini 2005), paraphrasing (Dolan, Quirk, and Brockett 2004), and more recently for elaboration (McCarthy and McNamara 2009).

Paraphrase identification in particular has been explored in the past by many researchers, especially after the release of the MSR Paraphrase Corpus (Dolan, Quirk, and Brockett 2004). We describe in this section several previous studies that are most related to our approach and leave others out, e.g., (Wu 2005; Kozareva and Montoyo 2006) due to space reasons.

Previous attempts to address the task of paraphrase identification range from simple to sophisticated. An example of a simple, yet quite accurate, approach is the one proposed by Zhang and Patrick (2005) who reported best results when using a small set of word substring overlap features (they used only four features) in combination with a decision tree learning method. In this work, they also described an ingenious solution to identify sentence-level paraphrase pairs by transforming source sentences into canonicalized text forms at the lexical and syntactic level, i.e., generic and simpler forms than the original text. One of the surprising findings is that the baseline system based on a supervised decision tree classifier with simple lexical matching features lead to better results when compared to the more sophisticated approaches that were experimented by them or others. They also revealed limitations of the MSR Paraphrase Corpus. The fact that their text canonicalization features did not lead to better than the baseline approach supports their findings that the sentential paraphrases, at least in the MSR corpus, share more words in common than one might expect given the standard definition of a paraphrase. The standard definition implies to use different words when paraphrasing. Zhang and Patrick used decision trees to classify the sentence pairs making their approach a supervised one as opposed to other approaches which are only *minimally supervised* (the most common of these types is when a similarity score is computed and then a threshold is learned from the training data, which will then be used to classify the instances).

The simple approach of Zhang and Patrick is slightly better than the best results reported by Mihalcea, Corley, and Strapparava (2006) who proposed an algorithm that extends word-to-word similarity metrics into a text-to-text semantic similarity metric based on which they decide whether two sentences are paraphrases or not. To obtain the semantic similarity between individual words, they used WordNet similarity package (Pedersen, Patwardhan, and Michelizzi 2004) which we will describe in more detail in chapter 3. Mihalcea and colleagues report best results when using an average over all word-to-word similarity metrics. It seems that one problem with Mihalcea and colleagues approach is the greedy strategy used to semantically match a word from one sentence to a word from the other sentence in a pair. They match the word with the maximum similarity among all the words in the second sentence.

Fernando and Stevenson (2008) on the other hand used the same basic idea of using word similarity metrics to identify paraphrases but instead of adopting a greedy strategy they opted for a more global strategy in which the similarity of all pairs of words is considered instead of just the maximum similarities. Fernando and Stevensons approach is called matrix similarity and provided significantly better results than Mihalcea and colleagues in terms of accuracy. However, (Mihalcea, Corley, and Strapparava 2006) had better recall results.

Rus and colleagues (Rus et al. 2008a) addressed the task of paraphrase identification by computing the degree of subsumption at lexical and syntactic level between two sentences in a bidirectional manner: from Text A to Text B and from Text B to Text A. The approach relied on a unidirectional approach that was initially developed to recognize the sentence-to-sentence relation of entailment (Rus et al. 2008a). Rus and colleagues' approach only used similarity to decide paraphrasing, ignoring dissimilarities which could be important to the final decision. The similarity was computed as a weighted sum of lexical matching, i.e., direct matching of words enhanced with synonymy information from WordNet, and syntactic matching, i.e., dependency overlap. Dependencies were derived from a phrase-based parser which outputs the major phrases in a sentence and organizes them hierarchically into a parse tree.

Another example of a sophisticated approach is the one recently proposed by Das and Smith (Das and Smith 2009) which employs a probabilistic approach that uses both syntactic and lexical semantics information to decide whether two sentences are paraphrases or not.

From one perspective, the above methods can be classified as similarity-centered, dissimilarity centered, or a combination of similarity and dissimilarity approaches. The similarity-centered (Mihalcea, Corley, and Strapparava 2006; Fernando and Stevenson 2008; Das and Smith 2009) focus on how similar two sentences are, based on which a similarity score is computed which is then used to make a decision: paraphrase or not. On the other hand, other researchers observed that sentential paraphrases in general have a high degree of lexical overlap and thus decided to focus on dissimilarities between sentences in a pair.

Qiu and colleagues (Qiu, Kan, and Chua 2006) proposed a two-phase architecture for paraphrase identification. In the first phase, they identified similarities between two sentences, while in the second phase the dissimilarities were classified with respect to their relevance in deciding the presence of paraphrase. Their approach uses predicate argument tuples that capture both lexical and syntactic dependencies among words to find similarities between sentences. The first phase is similar to our approach for detecting common dependencies. In the second phase, they used a supervised classifier to detect whether the dissimilarities are important.

A previous work which defines its research problem in terms very similar to ours is the work of Li and colleagues (2006). Like us, they propose a method to measure the semantic similarity between very short texts, such as sentences, by using information from structure lexical databases and from corpus statistics. The input sentences are organized as ordered lists of words which are compared at word level, based on word-to-word semantic similarity measures extracted from WordNet (Miller 1995). For every word in the first sentence, the most semantically similar word is searched in the second sentence and if the similarity score is above a certain threshold, then the words are paired in a computed vector-based representation. This representation also takes into account the order of words in the sentences, which they consider to be a representative aspect of primary syntactic information. One drawback they report to their method is that is relies on single word-to-word matching, so examples which use multiple words to represent concept, as in *unmarried man* versus *bachelor* don’t work too well. At the time this research was done, there wasn’t any available corpus to evaluate the proposed method, so Li and colleagues created their own small set of testing data, consisting of 30 instances of paired definitions of words with similar meanings. The instances were manually labeled by multiple annotators on 0 to 4 fixed point scale (0 - not similar; 4 - similar) and the scores were then averaged. For evaluation, they report performance in terms of correlation (r = 0.816) and compared it the average agreement scores between annotators (r = 0.825).

As in Corley and Mihalcea (Corley and Mihalcea 2005), Li and colleagues' method quantifies the degree of similarity between the inputs, while the dissimilarity aspect is not emphasized enough. Looking only at the similarity factor is common when working with larger texts where, for most of the time, to purpose is to search for semantically similar documents and less to evaluate whether two documents are semantically similar or not. On shorter texts however, we believe it is crucial to study the semantic small differences between the inputs, where for example the simple presence of a negation can significantly change the meaning of a sentence.

To summarize some of this previous work and compare it with our current work done on the MSR corpus, Table 7.1 presents best results reported by other researchers with their main approaches. We do not report results on incomplete data sets such as the ones reported by (Wan et al. 2006) who eliminated several hundred instances from both training and testing due to a technical problem they encountered with the syntactic parser they used. From the table, we observe that most of our methods presented in this dissertation provide competitive results in terms of accuracy, precision and recall. Increasing precision for paraphrase identification on the MSR corpus seems to be more challenging than obtaining high recall. Extremely high recall, in the upper 90s, can be easily obtained with

Table 7.1
Performance results of previous work done on MSR

	Acc.	Prec.	Recall
Corley & Mihalcea, 2005	71.50	72.30	92.50
Zhang & Patrick, 2005	71.90	74.30	88.20
Zhang & Patrick (baseline)	72.30	78.80	79.80
Mihalcea et al., 2006	70.30	69.60	97.70
Qui et al., 2006	72.00	72.50	93.40
Lintean & Rus, 2009	72.06	74.04	89.28
Fernando & Stevenson, 2008	74.10	75.20	91.30
Das & Smith, 2009	73.86	79.57	86.05
Wan et al. (repl. in Das&Smith)	75.42	76.88	90.14

very simple lexical overlap methods enhanced with lexical semantics (Mihalcea, Corley, and Strapparava 2006; Wan et al. 2006).

7.2. Future Directions

On several ocasions in previous chapters, where various methods of semantic similarity assessment were presented, we suggested a few interesting ideas on how our current research can be improved. This section reviews and summarizes some of these ideas.

Throughout the last four chapters, we presented methods of semantic similarity assessment of many varieties and many options and parameters to choose from. We did not experiment with many of the possible variants on these methods, and future research might explore all this space of possibilities and might find some variants of the proposed methods that are actually more accurate than the ones tested and presented in this current work.

In the fourth chapter, where we present methods based on word-to-word semantics, we suggest the possibility of how one can improve the pairing of words using a simple greedy approach, by using some methods of optimal matching. Unfortunately, these latter methods depend strongly on the word-to-word measures that are being used, which in some cases are not sufficient to find the correct optimal match. This suggest that when matching words between two texts, one should also look at the context in which the words are being used (i.e., what are their neighboring words, or which words do they syntactically relate to). In the same chapter, section 4.8, we suggested another way for representing the meaning of words in vectorial structures instead of LSA, and that is through the Latent Dirichlet Allocation (LDA) approach. Since LDA offers a few extra benefits when it comes to topic analysis and word sense disambiguation, we would encourage it to be further explored within the context of semantic similarity assessment.

The idea for using kernels to measure text-to-text semantic similarity is promising. Kernel methods used in combination with SVM learning are very powerful for processing natural language due to their ability to assimilate any type of information to any specified degree of detail. Although, in case too much detail is allowed to be assimilated, there is a likely risk for overfitting the data, SVMs are well known for handling outliers fairly well. In Chapter 6 we introduced the theory of kernel methods and SVM learning and we presented some basic lexical kernels that rely on the lexical dissimilarities between the texts. For future work, we plan to do more experiments on the lexical kernels proposed and create some new, more complex kernels for the task of semantic similarity assessment. A strong limitation of our current kernels is that they cannot handle the asymmetric relations of semantic similarity. It would be a good idea to further research this problem and see if these kernels can be adapted so that they could also handle those types of asymmetric relations.

BIBLIOGRAPHY

References

Androutsopoulos, Ion and Prodromos Malakasiotis. 2010. A survey of paraphrasing and textual entailment methods. *Journal of Artificial Intelligence Research*, 38:135–187.

Anthony, Martin and Norman Biggs. 1995. Pac learning and neural networks. In Michael A. Arbib, editor, *The handbook of brain theory and neural networks*. MIT Press, Cambridge, MA, USA, pages 694–697.

Azevedo, Roger, Amy Witherspoon, Arthur C. Graesser, Danielle McNamara, Amber Chauncey, Emily Siler, Zhiquiang Cai, Vasile Rus, and Mihai Lintean. 2008. Metatutor: An adaptive hypermedia system for training and fostering self-regulated learning about complex science topics. In *Meeting of Society for Computers in Psychology*, Chicago, IL.

Baeza-Yates, Ricardo and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval*. ACM Press.

Banerjee, S. and T. Pedersen. 2003. Extended gloss overlaps as a measure of semantic relatedness. In *In Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 805–810.

Barzilay, Regina and Lillian Lee. 2003. Learning to paraphrase: An unsupervised approach using multiple-sequence alignment. In *In HLT-NAACL 2003: Main Proceedings*, pages 16–23.

Berry, Michael W., Susan T. Dumais, and Gavin W. O'Brien. 1995. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37:573–595.

Bies, Ann, Mark Ferguson, Karen Katz, and Robert MacIntyre. 1995. Bracketing guidelines for treebank ii style. Penn Treebank Project.

Blei, David M., Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

Brockett, Chris and William B. Dolan. 2005. Support vector machines for paraphrase identification and corpus construction. In *In Proceedings of the 3rd International Workshop on Paraphrasing*, pages 1–8.

Burges, Christopher J. C. 1998. A tutorial on support vector machines for pattern recognition. *Data Minning Knowledge Discovery*, 2:121–167, June.

Chang, Chih-Chung and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at: http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Collins, Michael. 1996. A new statistical parser based on bigram lexical dependencies. In *In Proceedings of the 34th Annual Meeting of the ACL*, Santa Cruz.

Corley, Courtney and Rada Mihalcea. 2005. Measuring the semantic similarity of texts. In *In Proceedings of the ACL Workshop on Empirical Modeling of Semantic Equivalence and Entailment*. Ann Arbor, MI.

Covington, Michael A. 1990. A dependency parser for variable-word-order languages. Technical report, University of Georgia, Athens, January.

Cristianini, Nello and John Shawe-Taylor. 2000. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press.

Dagan, Ido, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *In Proceedings of the PASCAL Challenge Workshop on Recognizing Textual Entailment*.

Das, Dipanjan and Noah A. Smith. 2009. Paraphrase identification as probabilistic quasi-synchronous recognition. In *In Proceedings of the Joint Conference of the Annual Meeting of the ACL and the International Joint Conference on NLP*, Singapore, August.

de Marneffe, Marie-Catherine, Bill MacCartney, and Christopher D. Manning. 1993. Generating typed dependency parses from phrase structure parses. In *In Proceedings of LREC-06*, pages 449–454.

Dessus, Philippe. 2009. An overview of lsa-based systems for supporting learning and teaching. In *Proceeding of the 2009 conference on Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling*, pages 157–164, Amsterdam, The Netherlands, The Netherlands. IOS Press.

Doddington, George. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, HLT '02, pages 138–145, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Dolan, Bill, Chris Quirk, and Chris Brockett. 2004. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *In Proceedings of 20th International Conference on Computational Linguistics (COLING)*.

Dumais, Susan T. 1991. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments and Computers*, 23:229–236.

Fernando, Samuel and Mark Stevenson. 2008. A semantic similarity approach to paraphrase detection. In *In Proceedings of the Computational Linguistics UK (CLUK 2008)*.

Finch, Andrew, Young Sook Hwang, and Eiichiro Sumita. 2005. Using machine translation evaluation techniques to determine sentence-level semantic equivalence. In *In Proceedings of the 3rd International Workshop on Paraphrasing (IWP2005)*.

Graesser, Arthur C., Andrew Olney, Brian C. Haynes, and Patrick Chipman. 2005. Autotutor: A cognitive system that simulates a tutor that facilitates learning through mixed-initiative dialogue. In *In Cognitive Systems: Human Cognitive Models in Systems Design*. Mahwah: Erlbaum.

Graesser, Arthur C., Phanni Penumatsa, Matthew Ventura, Zhiqiang Cai, and Xiangen Hu, 2007. *Handbook of Latent Semantic Analysis*, chapter Using LSA in AutoTutor: Learning through Mixed-initiative Dialogue in Natural Language, pages 243–262. Lawrence Erlbaum Associates.

Harabagiu, Sanda, A Harabagiu, Dan Moldovan, Marius Pasaca, Rada Mihalcea, Mihai Surdeanu, Razvan Bunescu, Roxana Girju, Vasile Rus, and Paul Morarescu. 2000. Falcon: Boosting knowledge for answer engines. In *In Proceedings of The Ninth Text REtrieval Conference (TREC 9)*, pages 479–488.

Hays, David G. 1964. Dependency theory: A formalism and some observations. *Languages*, 40:511–525.

Heilman, Michael and Noah A. Smith. 2010. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *In Proceedings of the NAACL/HLT)*, Los Angeles, US.

Hirst, Graeme and David St-Onge. 1998. Lexical chains as representations of context for the detection and correction of malapropisms. *WordNet: An electronic lexical database*.

Ibrahim, Ali, Boris Katz, and Jimmy Lin. 2003. Extracting structural paraphrases from aligned monolingual corpora. In *In Proceedings of the ACL Workshop on Paraphrasing*, pages 57–64.

Iordanskaja, Lidija, Richard Kittredge, and Alain Polguere, 1991. *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, chapter Lexical selection and paraphrase in a meaning-text generation model, pages 293–312. Kluwer Academic Publishers, Norwell, MA, USA.

Jiang, Jay J. and David W. Conrath. 1997. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proc. of the Int'l. Conf. on Research in Computational Linguistics*, pages 19–33.

Jing, Liping, Lixin Zhou, Michael K. Ng, and Joshua Zhexue Huang. 2006. Ontology-based distance measure for text clustering. In *In Proceedings of the Fourth Workshop on Text Mining (SIAM-06*, Bethesda, Maryland.

Joachims, Thorsten, 1999. *Advances in Kernel Methods - Support Vector Learning*, chapter Making large-scale SVM learning practical. MIT Press.

Jurafsky, Daniel and James H. Martin. 2002. *Speech and Language Processing*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2nd edition edition, May.

Kate, Rohit J. 2008. A dependency-based word subsequence kernel. In *In Proceedings of EMNLP*.

Kozareva, Zornitsa and Andrs Montoyo, 2006. *Advances in Natural Language Processing: Lecture Notes in Computer Science*, volume 4139, chapter Paraphrase Identification on the basis of Supervised Machine Learning Techniques, pages 524–533. Springer-Verlag Berlin Heilderberg.

Kuhn, H. W. 2005. The hungarian method for the assignment problem. *Naval Research Logistics*, 52:7–21.

Landauer, Thomas K., Danielle S. McNamara, Simon Dennis, and Walter Kintsch. 2007. *Handbook of Latent Semantic Analysis*. Mahwah, NJ: Erlbaum.

Leacock, Claudia and Martin Chodorow, 1998. *WordNet, An Electronic Lexical Database*, chapter Combining local context and WordNet sense similarity for word sense identification. The MIT Press.

Li, Yuhua, David McLean, Zuhair A. Bandajar, James D. O'Shea, and Keeley Crockett. 2006. Sentence similarity based on semantic nets and corpus statistics. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1138–1150.

Lin, Dekang. 1993. Principle-based parsing without overgeneration. In *In Proceedings of ACL*.

Lin, Dekang. 1998. An information-theoretic definition of similarity. In *In Proceedings of the 15th International Conference on Machine Learning*, Madison, WI.

Lintean, Mihai and Vasile Rus. 2011. Dissimilarity kernels for paraphrase identification. In *In proceedings of Twenty-Fourth International FLAIRS Conference*, Palm Beach, FL.

Lintean, Mihai, Vasile Rus, and Arthur C. Graesser. 2008. Using dependency relations to decide paraphrasing. In *In Proceedings of the Society for Text and Discourse Conference*.

Lodhi, Huma, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444.

Madnani, Nitin and Bonnie J. Dorr. 2010. Generating phrasal and sentential paraphrases: A survey of data-driven methods. *Computational Linguistics*, 36.

Malakasiotis, Prodromos. 2009. Paraphrase recognition using machine learning to combine similarity measures. In *In Proceedings of the ACL-IJCNLP*, Suntec, Singapore, August.

Manning, Christopher D. and Heinrich Schutze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.

Martin, Dian I. and Michael W. Berry, 2007. *Handbook of latent semantic analysis*, chapter Mathematical foundations behind latent semantic analysis, pages 35–55. Lawrence Erlbaum Associates.

McCarthy, Philip M. and Danielle S. McNamara. 2009. User-language paraphrase corpus challenge. Online at https://umdrive.memphis.edu/pmmccrth/public/Paraphrase Corpus/Paraphrase_site.htm.

McCarthy, Philip M., Vasile Rus, Scott A. Crossley, Arthur C. Graesser, and Danielle S. McNamara. 2008. Assessing forward-, reverse-, and average-entailer indices on natural language input from the intelligent tutoring system, istart. In *In proceedings of Twenty-First International FLAIRS Conference*.

McKeown, Kathleen R. 1983. Paraphrasing questions using given and new information. *Comput. Linguist.*, 9:1–10, January.

McNamara, Danielle S., Chutima Boonthum, and Keith Millis, 2007. *Handbook of Latent Semantic Analysis.*, chapter Evaluating self-explanations in iSTART: comparing word-based and LSA algorithms, pages 227–241. Lawrence Erlbaum Associates.

McNamara, Danielle S., Irwin B. Levinstein, and Chutima Boonthum. 2004. istart: interactive strategy training for active reading and thinking. *Behavioral Research Methods, Instruments, and Computers*, 36(2):222–33.

Mihalcea, Rada, Courtney Corley, and Carlo Strapparava. 2006. Corpus-based and knowledge-based measures of text semantic similarity. In *In Proceedings of the American Association for Artificial Intelligence*. Boston, July.

Miller, George A. 1995. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41.

Moschitti, Alessandro. 2009. Syntactic and semantic kernels for short text pair categorization. In *In Proceedings of the 12th Conference of EACL*, pages 576–584, Athens, Greece.

Nakov, Preslav, Antonia Popova, and Plamen Mateev. 2001. Weight functions impact on lsa performance. In *In Proceedings of the EuroConference Recent Advances in Natural Language Processing*, pages 187–193.

Palmer, Martha, Paul Kingsbury, and Daniel Gildea. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei jing Zhu. 2001. Bleu: a method for automatic evaluation of machine translation. In *Meeting of the Association for*

Computational Linguistics.

Park, Eui-Kyu, Dong-Yul Ra, and Myung-Gil Jang. 2005. Techniques for improving web retrieval effectiveness. *Information Processing and Management*, 41(5):1207–1223.

Patwardhan, Siddharth. 2003. Incorporating Dictionary and Corpus Information into a Context Vector Measure of Semantic Relatedness. Master's thesis, University of Minnesota, Duluth, August.

Patwardhan, Siddharth, Satanjeev Banerjee, and Ted Pedersen. 2003. Using measures of semantic relatedness for word sense disambiguation. In *In Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics (CICLING-03*, pages 241–257.

Pedersen, Ted, Siddharth Patwardhan, and Jason Michelizzi. 2004. Wordnet::similarity - measuring the relatedness of concepts. In *In Proceedings of Fifth Annual Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-2004)*, pages 38–41, Boston.

Porter, M. F., 1997. *An algorithm for suffix stripping*, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Qiu, Long, Min-Yen Kan, and Tat-Seng Chua. 2006. Paraphrase recognition via dissimilarity significance classification. In *In Proceedings of EMNLP*, pages 18–26.

Ramage, Daniel, Anna N. Rafferty, and Christopher D. Manning. 2009. Randomwalks for text semantic similarity. In *In Proceedings of the 2009 Workshop on Graph-based Methods for Natural Language Processing*, Suntec, Singapore.

Resnik, Philip. 1995. Using information content to evaluate semantic similarity in a taxonomy. In *In Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 448–453.

Rinaldi, Fabio, James Dowdall, Kaarel Kaljurand, and Michael Hess. 2003. Exploiting paraphrases in a question answering system. In *In Proceedings of the 2nd International Workshop in Paraphrasing*, pages 25–32. Saporo, Japan.

Ruppenhofer, Josef, Michael Ellsworth, Miriam R. L. Petruck, Christopher R. Johnson, and Jan Scheffczyk. 2005. FrameNet II: Extended theory and practice. Technical report, ICSI.

Rus, Vasile and Arthur C. Graesser. 2007. Lexico-syntactic subsumption for textual entailnment. *In Recent Advances in Natural Language Processing IV: Selected Papers from RANLP 2005*.

Rus, Vasile, Mihai Lintean, Sajjan Shiva, and Darko Marinov. 2010. Automatic detection of duplicate bug reports using word semantics. In *In Proceedings of the 7th IEEE Working Conference on Mining Software Repositories*. Cape Town, South Africa.

Rus, Vasile, Philip M. McCarthy, Mihai C. Lintean, Danielle S. McNamara, and Arthur C. Graesser. 2008a. Paraphrase identification with lexico-syntactic graph subsumption. In *In proceedings of Twenty-First International FLAIRS Conference*.

Rus, Vasile, Philip M. McCarthy, Danielle S. McNamara, and Arthur C. Graesser. 2008b. A study of textual entailment. *International Journal on Artificial Intelligence Tools*, 17(4):659–685.

Seddah, Djame, Grzegorz Chrupala, Ozlem Cetinoglu, Josef van Genabith, and Marie Candito. 2010. Lemmatization and lexicalized statistical parsing of morphologically rich languages: the case of french. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, SPMRL '10, pages 85–93, Morristown, NJ, USA. Association for Computational Linguistics.

Shawe-Taylor, John, Peter L. Bartlett, Robert C. Williamson, and Martin Anthony. 1996. A framework for structural risk minimisation. In *Proceedings of the ninth annual conference on Computational learning theory*, COLT '96, pages 68–76, New York, NY, USA. ACM.

Vapnik, Vladimir N. 1998. *Statistical Learning Theory*. Wiley-Interscience, September.

Wan, Stephen, Mark Dras, Roberd Dale, and Cecile Paris. 2006. Using dependency-based features to take the para-farce out of paraphrase. In *In Proceedings of ALTW*.

Webster, Jonathan J. and Chunyu Kit. 1992. Tokenization as the initial phase in nlp. In *Proceedings of the 14th conference on Computational linguistics - Volume 4*, pages 1106–1110, Morristown, NJ, USA. Association for Computational Linguistics.

Weeds, Julie, David Weir, and Bill Keller. 2005. The distributional similarity of sub-parses. In *In Proceedings of the ACL Workshop on Empirical Modeling of Semantic Equivalence and Entailment*, pages 7–12, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Witten, Ian H. and Eibe Frank. 2005. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition edition.

Wu, Dekai. 2005. Recognizing paraphrases and textual entailment using inversion transduction grammars. In *In Proceedings of the ACL Workshop on Empirical Modeling of Semantic Equivalence and Entailment*, pages 25–30. Ann Arbor.

Wu, Zhibiao and Martha Stone Palmer. 1994. Verb semantics and lexical selection. In *In Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 133–138.

Zanzotto, Fabio Massimo, Marco Pannacchiotti, and Alessandro Moschitti. 2009. A machine learning approach to textual entailment recognition. *Natural Language Engineering*, 15(4):551–582.

Zhang, Yitao and Jon Patrick. 2005. Paraphrase identification by text canonicalization. In *In Proceedings of the Australasian Language Technology Workshop*.