# INVESTIGATION OF THE PROBABILISTIC LEXICAL ENTAILMENT MODEL FOR RECOGNISING TEXTUAL ENTAILMENT

Anne O'Donnell

Supervised by Prof. Stephen Pulman

MSc Dissertation

Oxford University Computing Laboratory

7 September, 2007

# Abstract

Multiple natural language processing applications, including information retrieval, question answering, information extraction, and document summarisation, involve the task of identifying whether or not the meaning of a given sentence (called the *hypothesis*) can be inferred from the truth of another similar sentence (called the *text*). This task is called *recognising textual entailment*. Another simpler sub-task, called *recognising lexical entailment*, involves determining whether every lexical concept denoted by a word in the hypothesis is entailed by a lexical concept contained in the text. The Bar Ilan probabilistic lexical entailment model, which was previously proposed for the textual entailment recognition task, aims to recognise textual entailment by solving the simpler lexical entailment recognition sub-task and using the lexical entailment relationship as an estimator for textual entailment. This paper further examines the probabilistic lexical entailment model as a predictor of textual entailment.

We attempt to improve the Bar Ilan model's performance on both the lexical and textual entailment recognition tasks by replacing its mechanism for recognising the lexical entailment probability (LEP) between words with two more accurate LEP metrics. The results show that while improving the accuracy of the LEP metric increased the model's accuracy on the lexical entailment recognition sub-task from 64% to 79.5%, it did not produce a corresponding increase in textual entailment recognition accuracy. This suggests that while there are a few adjustments which could be made to the Bar Ilan model to improve its lexical entailment recognition accuracy even further, greater gains in textual entailment recognition may come from combining the probabilistic lexical entailment model with other methods of semantic analysis.

# Acknowledgements

I would like to express my deep appreciation to the Marshall Commission for providing me with the tremendous opportunity to study for two years in the UK.

I would also like to thank Professor Stephen Pulman for introducing me to the field of computational linguistics and supervising my work in this project.

Last but not least, to my family and Will – my best judges – thank you for your endless support.

# Contents

Abstract ..... iii

Acknowledgements ..... v

Contents ..... vii

List of Figures ..... ix

Chapter One: Introduction ..... 1

1.1 The Textual Entailment Recognition task ..... 1

1.2 Objectives ..... 4

1.3 Structure ..... 5

Chapter Two: The Probabilistic Lexical Entailment Model ..... 7

2.1 The Probabilistic Setting ..... 7

2.2 Lexical Reference ..... 9

2.3 Limits of the Model ..... 11

2.4 Implementation ..... 12

2.4.1 Datasets ..... 12

2.4.2 Calculating $P(Tr_h = 1 \mid t)$ ..... 12

2.4.3 Calculating $P(Tr_h = 1)$ ..... 15

Chapter Three: Lexical Entailment between Pairs of Words ..... 16

3.1 Lexical Entailment Probability Metrics ..... 17

3.1.1 Web-based Co-occurrence Metric ..... 17

3.1.2 Syntactic Feature Distributional Similarity ..... 19

3.1.3 Thesaurus-based Similarity ..... 25

3.2 Comparison of LEP Metrics ..... 26

3.2.1 Judgement Criteria ..... 26

3.2.2 Testing $LEP_{co}$ ..... 27

3.2.3 Testing $LEP_{ws}$ ..... 28
3.2.4 Testing $LEP_{thes}$ ..... 31
3.2.5 Results ..... 31
3.2.4 Converting Similarity Metrics to Lexical Entailment Probabilities ..... 33
Chapter Four: Embedding the New Lexical Entailment Probability Metrics ..... 35
4.1 Smoothing ..... 35
4.2 Textual Entailment Recognition Accuracy ..... 36
4.3 Lexical Entailment Recognition Accuracy ..... 41
4.4 Discussion ..... 46
Chapter Five: Conclusion ..... 55
Appendix A: Stop Words ..... 57
Appendix B: Pseudocode for the Textual Entailment Model ..... 59
Appendix C: Pseudocode for calculating $LEP_{co}(u,v)$ ..... 61
Appendix D: Java code for counting word-feature pairs ..... 63
Appendix E: Java code for calculating feature weights ..... 71
Appendix F: Java code for calculating $LEP_{ws}$ rankings ..... 75
Bibliography ..... 85

# List of Figures

Figure 1. Textual entailment examples taken from the first Recognising Textual Entailment Challenge dataset. ..... 3

Figure 2. Example of input data format taken from the RTE-1 development dataset. ..... 13

Figure 3. Example of output data format containing results of processing the input in Figure 2. ..... 15

Figure 4. Example of Minipar grammatical relationship triplet. ..... 20

Figure 5. Pseudocode algorithm for counting word-feature pairs. ..... 23

Figure 6. Pseudocode explaining implementation of LEPws(u,v) ..... 25

Figure 7. Diagram depicting the relationship between n, F(n), R(n), and WS(f). ..... 30

Figure 8. Pseudocode description of algorithm used to calculate top-40 words for random nouns as evaluated by $LEP_{ws}$. ..... 30

Figure 9. Top 20 most probably entailed or entailing words for the noun policy for each LEP metric. ..... 31

Figure 10. Precision values for Top-10/20/30/40 words as judged by each of three LEP metrics. ..... 32

Figure 11. Confusion matrix for two judges' assessment of a sample of word pairs. ..... 33

Figure 12. Calculated cut-off values and performance of the Bar Ilan model on the textual entailment recognition task, using the RTE-1 development dataset (567 sentence pairs). ..... 37

Figure 13. Performance of the Bar Ilan model on the textual entailment recognition task; RTE-1 test dataset (800 sentence pairs). ..... 38

Figure 14. Textual Entailment Similarity Score v Similarity Score. ..... 40

Figure 15. Performance of the Bar Ilan model on the lexical entailment recognition task, using the author-annotated dataset of 200 sentence pairs, with testing done using 10-fold cross validation. ..... 42

Figure 16. Alignment for sentence pair (ID=1251) produced by each LEP metric. ..... 44

Figure 17. Alignment for sentence pair (ID=1250) produced by each LEP metric. ..... 45

Figure 18. Alignment for sentence pair (ID=211) produced the $LEP_{thes}$ metric and manual alignment. ..... 46

Figure 19. Percentage of sentence pairs correctly classified by model. The precision scores are taken from Figure 10. ..... 47

Figure 20. Harmonic mean by model. ..... 48

Figure 21. Distribution of similarity scores for lexical (top) and textual (bottom) entailment datasets.. ..... 50

Figure 22. Comparison of lexical and textual entailment datasets. ..... 51

Figure 23. Example of sentence pair that is lexically but not textually entailed. ..... 51

Figure 24. Examples of sentence pairs which are textually but not lexically entailed. ..... 52

# Chapter One: Introduction

## 1.1 The Textual Entailment Recognition task

Human language is rich enough to express a single concept in countless ways. One of the major challenges in getting computers to summarise, answer questions about, or extract information from natural language text is to enable the computer to recognise when two different pieces of text convey the same meaning. For example, a multi-document summarisation system might attempt to summarise two documents containing the sentences below[1]:

> **Document 1:** "*The International Atomic Energy Agency report detailing the discovery also faulted Tehran for not cooperating with the U.N. watchdog's attempts to investigate other suspicious aspects of Iran's nuclear programme.*"
>
> **Document 2:** "*Tehran did not cooperate with the U.N. watchdog's attempts to investigate suspicious aspects of Iran's nuclear programme.*"

The system must recognise that the two sentences convey the same concept and include this concept in its final summary. Additionally, the system should be able to tell that

[1] Sentence examples taken from the Third Recognising Textual Entailment Challenge dataset, available from http://www.pascal-network.org/Challenges/RTE3/.

the first sentence entails the second and therefore include the entailed sentence in the summary to prevent redundancy.

Until a few years ago, this subtask of determining whether one textual statement could be inferred from another text fragment was addressed separately for each natural language processing application (Glickman, 2006). A researcher attempting to judge the performance of this subtask in a Question Answering system, for example, could only do so by testing his subsystem against others that had been developed for Question Answering systems. Furthermore, advances in this subtask were rarely translated across applications. For these reasons, the PASCAL Recognizing Textual Entailment (RTE) Challenge was introduced in 2004 to introduce *textual entailment recognition* as a generic applied and evaluated task for natural language processing researchers (Dagan, Glickman, & Magnini, 2006).

The RTE task definition defines *textual entailment* as a directional relation between two text fragments called the *text* (the entailing text) and the *hypothesis* (the entailed text) as follows (Bar-Haim, et al., 2006):

**Definition 1:** We say that a text *T* **textually entails** a hypothesis *H* if, typically, a human reading *T* would infer that *H* is most likely true.

This operational definition assumes common background knowledge and common human understanding of language. The definition is quite informal because it reflects the guidelines given to human annotators when manually judging entailment between pairs of sentences, a task which is somewhat uncertain given the variability of language. Nevertheless it has been shown that humans achieve a fairly high degree of agreement in judging entailment between sentence pairs. The report on the Second Recognising Textual Entailment Challenge (RTE-2) (Bar-Haim, et al., 2006) says that in creating the RTE-2 dataset, the average agreement between each pair of manual annotators who shared at least 100 examples was 89.2%. The average Kappa score

between pairs of annotators was 0.78, indicating "substantial agreement" (Landis & Koch, 1997).

| Example | Text | Hypothesis | Entailment |
|---|---|---|---|
| 1 | Mexico City has a very bad pollution problem because the mountains around the city act as walls and block in dust and smog. | Poor air circulation out of the mountain-walled Mexico City aggravates pollution. | TRUE |
| 2 | LaFarge was the one who helped Tiffany to make the Favrile glass, said auctioneer William Doyle who operates an auction house in Manhattan. | William Doyle lives in Manhattan. | FALSE |
| 3 | More than 250 paintings commemorate the centennial of the Man Ray's birth in Philadelphia. | Man Ray was born in Philadelphia. | TRUE |
| 4 | The government announced last week that it plans to raise oil prices. | Oil prices drop. | FALSE |
| 5 | The Philippines has begun pulling its troops out of Iraq, a move seemingly being made to satisfy demands by kidnappers of a Filipino hostage. | Filipino soldiers are leaving Iraq. | TRUE |

***Figure 1.*** *Textual entailment examples taken from the first Recognising Textual Entailment Challenge dataset.*

Figure 1 provides a few examples of text and hypothesis pairs developed for the first Recognising Textual Entailment Challenge (RTE-1) dataset.[2] The participants in this challenge were provided with a development set of 567 sentence pairs and a test set of 800 sentence pairs. They came up with textual entailment recognition systems ranging in accuracy from 49.5% to 58.6% on the entire dataset (Glickman, 2006). The methods they used for judging textual entailment included semantic reasoning (Tatu &

[2] http://www.pascal-network.org/Challenges/RTE/

Moldovan, 2005), syntactic matching (de Salvo Braz, Girju, Punyakanok, Roth, & Sammons, 2005), and word overlap (Perez & Alfonseca, 2005) among others.

One of the best scoring systems at RTE-1 was developed by a group from Bar Ilan University and used a probabilistic approach to solve the simpler sub-problem of *lexical entailment* (Glickman, 2006). Lexical entailment recognition – the process of determining whether a lexical concept represented by a word or phrase in the hypothesis is entailed by a word or words in the text – is a necessary, but not sufficient, criterion for textual entailment to exist (Glickman, Dagan, & Koppel, A Probabilistic Classificaiton Approach for Lexical Textual Entailment, 2005). The Bar Ilan model attempted to establish whether lexical entailment held between the hypothesis and text and used this result as an estimate for the existence of a textual entailment relationship between the two. It did so assuming a generative probabilistic model, similar to the generative models used successfully in other areas of natural language processing such as machine translation (Brown, et al., 1990). The overall goal of this project is to further examine the probabilistic lexical entailment approach to recognizing textual entailment by attempting to improve the Bar Ilan model from RTE-1.

## 1.2 Objectives

The strong performance of the Bar-Ilan model at RTE1 suggests that the probabilistic lexical entailment approach holds promise for predicting textual entailment and warrants further investigation. The main goal of this project was to improve the accuracy of the Bar Ilan model by building on the work of (Glickman, 2006).

The Bar Ilan model created an alignment between words of the text and hypothesis such that each word in the hypothesis aligned with the word from the text which was most likely to have entailed it according to a lexical entailment probability metric. The Bar Ilan model used a simple metric based on word co-occurrence frequency on the web to evaluate the probability of lexical entailment between pairs of words. The main thrust of work in this project centred on implementing the probabilistic lexical entailment model with more accurate lexical entailment probability metrics in order to

see whether improving the accuracy of this subtask – recognising the lexical entailment relationship between pairs of words – would improve the entire model's accuracy in predicting lexical and textual entailment.

## 1.3 Structure

The rest of this paper is organised as follows. Chapter Two describes the probabilistic lexical entailment model in more detail. It gives an explanation of the probabilistic assumptions under which the Bar Ilan model operates and illustrates how the probabilistic model is used to predict the presence or absence of lexical entailment for an entire text and hypothesis pair. The chapter finishes with a description of the model's implementation for this project.

Chapter Three explains the three lexical entailment probability metrics that were implemented for this project – the original web-based co-occurrence metric from the Bar Ilan model, a metric based on distributional similarity of syntactic features, and a third metric based on a pre-existing thesaurus. The various metrics are judged and compared for accuracy.

Chapter Four describes the process of embedding each of the three lexical entailment probability metrics within the overall probabilistic model. We judge the performance of the system on the lexical and textual entailment tasks using each metric in order to see whether improved lexical entailment probability estimates lead to increased accuracy of the overall model. The results show that replacing the simple web-based co-occurrence metric from the Bar Ilan model with the two more accurate metrics improves the model's accuracy on the lexical entailment task but not on the textual entailment task.

In Chapter Five we state the project's conclusions and make suggestions for further research.

# Chapter Two: The Probabilistic Lexical Entailment Model

One of the top performers at the first Recognising Textual Entailment Challenge was the system created by researchers at Bar Ilan University that estimated textual entailment by solving the simpler task of predicting *lexical reference* using a probabilistic model (Glickman & Dagan, 2005). This project implements the Bar Ilan model following the guidelines set out in that paper. Chapter Two paraphrases the descriptions of lexical reference and the underlying probabilistic model from (Glickman, 2006) as background for the reader. It finishes with an explanation of how this model was implemented for this project using the Java programming language.

## 2.1 The Probabilistic Setting

Recognising textual entailment can be viewed as a probabilistic task. For some text/hypothesis pairs, the presence or absence of a textual entailment relationship is certain. For example, given the text "*Kate went for a swim at 6:00 last night*" and the hypothesis "*Kate swam yesterday evening*," it is definite that the text entails the hypothesis because they convey the same information. In other words, we can say that the text entails the hypothesis with probability of 1. On the other hand, for the text

"*Kate did not swim last night*" and hypothesis "*Kate swam last night*," the probability that the text entails the hypothesis is 0 since the hypothesis negates the text. Other text/hypothesis pairs have a less certain entailment relationship. The text "*Kate goes swimming most evenings*" definitely increases the probability that the hypothesis "*Kate went swimming last night*" is true, but it does not entail it necessarily. We can say that the probability that this last text entails the hypothesis is somewhere between 0 and 1.

Because of the probabilistic nature of the textual entailment recognition task, the Bar Ilan model assumes a generative probabilistic setting. The full model is described formally in (Glickman, 2006), but the main features are conveyed here to provide background for the reader.

Let $T$ signify the space of possible texts, and $t \in T$ a specific text. Let $H$ denote the set of all possible hypotheses, and the hypothesis $h \in H$ a propositional statement which can be assigned a truth value of 0 or 1. The possible world $w$ signifies a mapping from $H$ to { 0=false, 1=true } and is denoted by $w : H \rightarrow \{0,1\}$. It represents a specific state of affairs with concrete truth assignments for all possible propositions $h \in H$. The set of all possible worlds is signified by $W$.

The probabilistic generative model assumes that "texts are generated *along with* a concrete state of affairs" [author emphasis]. That is, when a source generates a text $t$, it also generates the set of truth assignments for all propositions $h$ which relate to the text, and that set of truth assignments comprises the possible world $w$. The truth assignments in $w$ therefore do not reflect some abstract "real world" but only convey the truth or untruth of hypotheses $h$ relating to $t$. The probability of generating a truth assignment for a hypothesis $h$ that is not at all related to the text is some prior P($h$), and the probability of a given hypothesis $h$ being true is higher than the prior when the related text supports the truth of $h$. (Glickman, 2006, pp. 53-55)

Let the random variable $Tr_h$ signify the truth value given to the hypothesis $h$ in a given world $w$. We use the statement $Tr_h = 1$ to denote the event that $h$ is assigned the truth value of 1 or *true*. For a text $t$, the variable $t$ also denotes the event that the generated text is $t$. The definition of *probabilistic textual entailment* says that a text $t$

probabilistically entails a hypothesis $h$ if $t$ increases the likelihood of $h$ being true. Using $t \Rightarrow h$ to denote the event that $t$ probabilistically entails $h$, the formal definition is below (Glickman, 2006, p. 55):

**Definition 2:** For all $t \in T$ and $h \in H$, $t \Rightarrow h$ iff $P(Tr_h = 1 \mid t) > P(Tr_h = 1)$.

The Bar Ilan model uses this probabilistic setting as its basis for modelling textual entailment. This reduces the problem to estimating $P(Tr_h = 1)$ and $P(Tr_h = 1|t)$.

## 2.2 Lexical Reference

A common strategy in natural language processing is to simplify a complex task into a simpler subtask and to use the results of the subtask as an estimate for the result of the initial, complex task. The Bar Ilan model follows this strategy by breaking the task of textual entailment recognition into the easier subtask of *lexical reference* (Glickman, 2006, p. 40):

**Definition 3:** A word $W$ is **lexically referred** in a text $T$ if there is an explicit or implied reference in $T$ to a concept denoted by $W$.

The Bar Ilan lexical entailment model uses lexical entailment to estimate $P(Tr_h = 1|t)$. It assumes that a hypothesis $h$ is true if and only if all its lexical components are true, and assigns a truth value to each word $u$ in the hypothesis indicating the presence or absence of a lexical reference to $u$ within the given text.

The probability that a given word $u$ within the hypothesis is true is assumed to be independent of the probabilities of the other words being true, giving the following estimations for the probability of the truth of the complete hypothesis:

$$P(Tr_h = 1|t) = \prod_{u \in h} P(Tr_u = 1|t) \tag{2.1}$$

$$P(Tr_h = 1) = \prod_{u \in h} P(Tr_u = 1) \tag{2.2}$$

Furthermore, the model assumes an alignment between words of the hypothesis and words of the text such that each entailed hypothesis word is entailed by a specific word in the text. Using $T_v$ to denote the event that the word $v$ appears within the text, this can be written mathematically as:

$$P(Tr_u = 1|t) = \max_{v \in t} P(Tr_u = 1|T_v) \tag{2.3}$$

Thus, by combining (2.1) and (2.3), we can obtain an overall estimate for $P(Tr_h = 1|t)$:

$$P(Tr_h = 1|t) = \prod_{u \in h} \max_{v \in t} P(Tr_u = 1|T_v) \tag{2.4}$$

We denote as *LEP*($u$,$v$) the *lexical entailment probability* between words $u$ and $v$, which is the same thing as $P(Tr_u = 1|T_v)$:

$$LEP(u,v) = P(Tr_u = 1|T_v) \tag{2.5}$$

Combining Equation 2.4 and Equation 2.5 with Definition 2, we derive the complete *probabilistic lexical entailment model* as used by the Bar Ilan model and which was implemented for this project:

**Definition 4.** For all $t \in T$ and $h \in H$, the **probabilistic lexical entailment model** determines the lexical entailment relationship between $t$ and $h$ as

$$(t \Rightarrow h) \leftrightarrow \prod_{u \in h} \max_{v \in t} LEP(u,v) > P(Tr_h = 1)$$

The Bar Ilan model uses a simple co-occurrence frequency metric to estimate $LEP(u,v)$ and empirically tunes a single cut off value λ to estimate $P(Tr_h = 1)$.

## 2.3 Limits of the Model

Lexical reference is generally a prerequisite for textual entailment to hold, but a textual entailment relationship does not necessarily hold wherever a lexical entailment relationship is present. This is apparent if one considers the text "*John is not at home"* and the hypothesis "*John is at home*." The hypothesis is definitely lexically entailed by the text as every word in the hypothesis is also present in the text, but the meanings of the sentences are contradictory and therefore textual entailment is clearly false.

Nevertheless, lexical entailment is a fairly good predictor of the textual entailment relationship. This was shown by (Glickman, 2006), in which the relationship between lexical and textual entailment was investigated in detail. Manual annotators judged a

set of text and hypothesis pairs from the RTE-1 development dataset for lexical and textual entailment. The experiment showed that a system which judged lexical entailment perfectly would achieve 69% accuracy on the textual entailment task (with 67% precision and 69% recall); in other words, the lexical and textual relationships matched up in 69% of sentence pairs. Thus a system that implements the probabilistic lexical entailment model perfectly should only hope to be about 69% accurate in predicting textual entailment.

## 2.4 Implementation

For this project we implement the probabilistic lexical entailment model using the Java programming language and following the guidelines given in (Glickman, Dagan, & Koppel, 2005). The objective of the implemented model is to be able to determine, given a text sentence and a hypothesis sentence, whether the text entails the hypothesis.

### 2.4.1 Datasets

The RTE-1 organisers created a large dataset of text and hypothesis pairs for training and testing textual entailment recognition systems. Each pair has been manually judged for the textual entailment relationship. The dataset, available for download online,[3] includes a development set of 567 pairs (283 *TRUE* and 284 *FALSE* for entailment) and a test set of 800 pairs (400 *TRUE* and 400 *FALSE*). We use these datasets for training and testing our implementation of the Bar Ilan probabilistic lexical entailment model.

### 2.4.2 Calculating $P(Tr_h = 1 \mid t)$

The process of predicting the textual entailment relationship has two main steps. First, the program must estimate the probability that the hypothesis is true given the

[3] http://www.pascal-network.org/Challenges/RTE/Datasets/

existence of the text: $P(Tr_h = 1 \mid t)$. Then, to determine whether the text actually entails the hypothesis, the system must compare the resulting value to the prior probability that the hypothesis is true on its own: $P(Tr_h = 1)$.

The program receives as input an `.xml` file containing text and hypothesis sentences in the format specified by the RTE-1 datasets. In this format, text and hypothesis sentences are denoted by the tags `<t>` and `<h>` respectively. A `<pair>` tag indicates the values of the attributes `id` (a numerical value identifying the sentence pair), `value` (indicating the existence or non-existence of the textual entailment relationship, as judged by human annotators), and `task`.[4]

```
<pair id="1977" value="TRUE" task="PP">
      <t>His family has steadfastly denied the charges.</t>
      <h>The charges were denied by his family.</h>
</pair>
```

***Figure 2.*** *Example of input data format taken from the RTE-1 development dataset.*

The model's first intermediate task is to calculate the probability that the hypothesis is true given the text, $P(Tr_h = 1|t)$, as in Equation 2.4. We begin by removing the `<pair>`, `<t>`, and `<h>` tags and deleting leading and trailing white space. We then convert all digits to 0, remove punctuation characters, and convert everything to lowercase.

At this point we remove words that belong to a pre-determined stop list comprising the 50 most common words from our training corpus, a roughly 20 million token subset of the British National Corpus (BNC Consortium, 2001) (See Appendix A). The reasons for using a stop list are twofold. First, the web-based co-occurrence lexical entailment probability estimate that we will embed within the model tends to assign inappropriately high similarity scores to common words. Second, the words in the stop

---

[4] The RTE-1 dataset labels sentence pairs according to the text processing application that created them for comparison. The labels include Comparable Documents (CD), Machine Translation (MT), Information Extraction (IE), Reading Comprehension (RC), Paraphrase (PP), Information Retrieval (IR), and Question Answering (QA). This project did not consider task labels.

list are common determiners, auxiliary verbs, conjunctions, and prepositions that rarely have an impact on the overall meaning of the phrase; they do not make up the 'meat' of the sentences they are in. Finally, we split what remains of the text and hypothesis at white space and put each sentence into its own array.

Once the text $t$ and hypothesis $h$ are in arrays, we simply go through the hypothesis words $u \in h$ one at a time to find the value of $\max_{v \in t} LEP(u, v)$. For each hypothesis word $u$, we first check whether any word from the text $v \in t$ is equal to $u$. If this is the case we align the equivalent words and say that $\max_{v \in t} LEP(u, v) = 1$. If none of the words $v \in t$ matches $u$, then we find the word $v \in t$ that returns the highest value for $LEP(u, v)$. The function $LEP(u, v)$ implements one of the three lexical entailment probability metrics to be explained in Chapter 3. Once we calculate $\max_{v \in t} LEP(u, v)$ for each word $u \in h$, we multiply the results to obtain $P(Tr_h = 1|t)$. A pseudocode version of the algorithm is given in Appendix B.

The output of this program serves two purposes. First, for each sentence pair we must be able to extract the estimated value of $P(Tr_h = 1|t)$ along with the manually annotated value of the textual entailment relationship (*TRUE* or *FALSE*) in order to evaluate the accuracy of the system. Second, to aid in later analysis, the program should output the alignments produced between the text and hypothesis along with the individual values for $LEP(u, v)$. Figure 4 shows the output format, which is similar to the input format except that the text and hypothesis now contain only the words which appeared in the text and hypothesis arrays, and two new tags are added. The results tag `<r>` gives the alignment produced by the model in the format `<p> u v LEP(u,v) </p>` for each hypothesis word, where $u$ is a hypothesis word and $v$ is the text word aligned with $u$. The score tag `<s>` tells which lexical entailment probability model was used (web-based co-occurrence (CO), syntactic feature distributional similarity (WS), or thesaurus (THES)) and gives the estimate for $P(Tr_h = 1|t)$.

```
<pair id="1977" task="PP" value="TRUE">
      <t> family steadfastly denied charges </t>
      <h> charges denied family </h>
      <r>
            <p> charges charges 1.0 </p>
            <p> denied denied 1.0 </p>
            <p> family family 1.0 </p>
      </r>
      <s model="CO" score="1.0">
</pair>
```

***Figure 3.*** *Example of output data format containing results of processing the input in Figure 2.*

### 2.4.3 Calculating $P(Tr_h = 1)$

Once the program has calculated a value for the probability that the hypothesis is true given the text, $P(Tr_h = 1|t)$, it must compare this with $P(Tr_h = 1)$ to see whether the text actually increases the probability that the hypothesis is true in accordance with Equation 2.4. The Bar Ilan model does this by empirically tuning a value for $P(Tr_h = 1)$ and classifying only sentence pairs for which $P(Tr_h = 1|t) > P(Tr_h = 1)$ as entailing.

In order to empirically tune the value for $P(Tr_h = 1)$ we use the Weka data mining software (Witten & Frank, 2005) to run the C4.5 decision tree algorithm on the scores from our training dataset. The C4.5 classification algorithm (called J48 in Weka) takes in our data with two attributes – score for $P(Tr_h = 1|t)$ as calculated by our model and value of *TRUE* or *FALSE* as manually judged. By creating a decision tree with just one node, it determines the cut-off score λ which maximises the information gain when all pairs for which $P(Tr_h = 1|t) > \lambda$ are classified as *TRUE* and all others are classified as *FALSE*.

Finally, using $\lambda = P(Tr_h = 1)$, we can test the accuracy of our model in predicting textual entailment by running our model on the test set and classifying the results based on whether $P(Tr_h = 1|t) > P(Tr_h = 1)$.

# Chapter Three: Lexical Entailment between Pairs of Words

As mentioned earlier, the Bar Ilan model uses a very simple co-occurrence frequency metric to estimate the lexical entailment probability between pairs of words. The task of calculating similarity between words has already been researched as a separate subtask quite extensively, as in the papers (Lin, Automatic retrieval and clustering of similar words, 1998), (Geffet & Dagan, 2004), and (Geffet & Dagan, 2005). In those papers the words are compared based on their shared syntactic features rather than simply their co-occurrence. A third method of judging similarity between words is to use a pre-existing thesaurus. One might expect the two latter similarity models to be more accurate at picking substitutable words than the web-based co-occurrence frequency estimate, and indeed our experiments in this chapter show this to be the case. Here we give an explanation and comparison of the three LEP metrics and their implementation for this project. The next chapter will explain how each performed when embedded within the probabilistic lexical entailment model.

## 3.1 Lexical Entailment Probability Metrics

The lexical entailment probability (LEP) estimate is the piece of the probabilistic lexical entailment model which determines which (if any) word from the text most probably entails a given word from the hypothesis. The main goal of this project is to improve the Bar Ilan model's accuracy in predicting lexical and textual entailment by replacing its LEP estimate with two other, more accurate LEP metrics. This section describes the web-based co-occurrence frequency estimate from the Bar Ilan model as well as the syntactic feature distributional similarity and thesaurus-based similarity estimates that we implement for this project.

### 3.1.1 Web-based Co-occurrence Metric

The RTE-1 Bar Ilan model estimates the probability that one word entails another using simple document co-occurrence frequency counts from the web. The formula is:

$$LEP_{co}(u, v) = \frac{n_{u,v}}{n_v} \tag{3.1}$$

where $n_{u,v}$ represents the number of hits from a web search for '*u AND v*' and $n_v$ represents the number of hits from a web search for '*v*' (Glickman, 2006, p. 62).

This unsupervised estimate is based on the probabilistic model from Chapter 2.1, assuming that documents in a corpus (the web) are generated by a language source. The language source generates each text, or web page, along with the hidden possible world that constitutes truth assignments for propositions (hypotheses) about that text. The metric makes the simple assumption that words which are stated verbatim in the text are true, and all other words are false. Thus, the lexical entailment probability $LEP(u,v) = P(Tr_u = 1 \mid T_v)$ (2.5) is estimated as the probability that $u$ appears in a document given that the word $v$ appears:

$$P(Tr_u = 1|T_v) \approx P(T_u|T_v) \tag{3.2}$$

In practice this estimate is calculated using maximum likelihood counts from the corpus, deriving Equation 3.1 (Glickman, Dagan, & Koppel, 2005).

In order to implement this LEP metric for our project we use the Alexa Web Search Service[5] which allows us to submit and receive results from the Alexa web search engine from within a Java programme. A variety of other web search APIs exist but we use Alexa for this project because it does not place a limit on number of queries per day (unlike the Yahoo! Web Search Services which limit queries to 5000 per day[6]) and it enables us to extract the number of hits for a web search (unlike the Google AJAX Search API which limits the number of results returned). This flexibility comes with a cost, however; the Alexa Web Search Service, which is run by Amazon Web Services, charges $0.00030 per request. This makes it necessary to carefully limit the number of calls to the web search engine and to maintain a cache of previous queries to avoid duplicates.

To calculate $LEP_{co}(u, v)$ it is necessary to know the number of hits for '*v*' and the number of hits for '*u AND v*'. The implementation of the function for this project first checks the cache to see whether either query has already been made. Since '*u AND v*' returns the same number of hits as '*v AND u*', we always search for the pair of terms in alphabetical order. If either one of the results is not in the cache, we perform a web search and add the resulting hits to the cache. Finally we divide $n_{u,v}$ by $n_v$ to calculate $LEP_{co}(u, v)$. A pseudocode description of the algorithm is in Appendix C.

---

[5] http://aws.amazon.com/alexawebsearch

[6] The RTE-1 development and test datasets include 7,310 distinct words and 56,877 distinct word pairings, not including stop words. Processing these files alone would have taken nearly 13 days. Furthermore, calculating the accuracy of the co-occurrence metric required comparing each of 10 random nouns to an entire vocabulary which could only be limited to about 20,000 words; performing these queries would have taken 40 days.

### 3.1.2 Syntactic Feature Distributional Similarity

The second lexical entailment probability metric is a word similarity measure proposed in (Lin, 1998) and tested in (Geffet & Dagan, 2004) for its ability to predict lexical entailment. The metric is based on the distributional similarity scheme, following from the Harris distributional hypothesis (Harris, 1954) which says that words that occur within the same context tend to have similar meanings.

As in (Geffet & Dagan, 2004), the metric constructs a weighted feature vector to characterise each word *w*. Features, made up of a word with which *w* co-occurs and the grammatical relationship between them, are extracted from a corpus parsed for grammatical dependencies. Following the notation from (Lin, 1998), we represent grammatical dependencies from the parsed corpus by the triple (*w,r,w'*) where *r* stands for the grammatical relationship between *w* and *w'* (i.e. *obj, subj, nmod,det* etc) and we represent features by the triplet <*w,r, direction* > in which *direction* indicates whether *w* is the head word or dependent word in the grammatical relationship. Thus the dependency triple (*w,r,w'*) translates to two separate word-feature pairs denoted as (*w,f*). They are (*w*, <*w',r,D*>) and (*w'*, <*w,r,H*>).

Once word-feature pairs have been extracted from the corpus, the metric applies a weighting function to calculate the weight for each feature *f* within each word's vector. In this case we use the Mutual Information (MI) weighting function (Lin, 1998) (Dagan, 2000), defined by:

$$MI(w,f) = \log_2 \frac{P(w,f)}{P(w)P(f)} \tag{3.3}$$

where $P(w,f)$ gives the probability that a random word-feature pair picked from the corpus is ($w$, $f$), $P(w)$ gives the probability of $w$ occurring within the corpus, and $P(f)$ gives the probability of $f$ occurring in the corpus. If we denote as $|w,f|$ the frequency count of the word-feature pair ($w$, $f$) and use * as a wildcard, we can estimate the *MI* measure using frequency counts from the parsed corpus as follows:

$$MI(w,<v,r,D>) = \log_2 \frac{\frac{|w,<v,r,D>|}{|*,*|}}{\frac{|w,*|}{|*,*|} \times \frac{|*,<v,r,D>|}{|*,*|}} = \log_2 \frac{|w,<v,r,D>| \times |*,*|}{|w,*| \times |*,<v,r,D>|}$$

(3.4)

Once weighted feature vectors have been constructed for each word, the syntactic feature distributional similarity metric calculates the similarity between words by applying a similarity function to the words' vectors. Once again this project adopts the similarity function used in both (Lin, 1998) and (Geffet & Dagan, 2004), where $F(w)$ denotes the set of *active features* within the feature vector of $w$:

$$sim(w,v) = \frac{\sum_{f \in F(w) \cap F(v)} MI(w,f) + MI(v,f)}{\sum_{f \in F(w)} MI(w,f) + \sum_{f \in F(v)} MI(v,f)}$$

(3.5)

We use $sim(w,v)$ directly as the lexical entailment probability estimate, assuming that if two words share all the same features, they will lexically entail each other:

$$LEP_{ws} = sim(w,v)$$

(3.6)

Calculating the syntactic feature distributional similarity between two words is a four step process. The first step is to actually parse the corpus with a parser that can extract dependency relationships. For this project we use the Minipar[7] dependency parser on a portion of the British National Corpus (BNC Consortium, 2001) consisting of roughly 20 million tokens. Minipar outputs grammatical relationship triplets one per line as below:

```
culture     N:mod:A     popular
```

***Figure 4.*** *Example of Minipar grammatical relationship triplet.*

[7] http://www.cs.ualberta.ca/~lindek/minipar.htm

In this implementation, the BNC is split into thousands of files, and the Minipar parser outputs one parsed file for each BNC input file.

Once we have the corpus parsed for grammatical relationships, the second step is to extract and count the word-feature pairs from the grammatical relationship triplets produced by the parser. Performing this procedure the naïve way, by maintaining a count for each encountered word-feature pair in memory, would require a very large amount of RAM. As a rough estimation, consider that our parsed corpus produced nearly 10 million distinct word-feature pairs. If we store an integer for the count of each word-feature pair, and an integer takes 16 bytes of memory in Java, this means that maintaining the counts alone for each word-feature pair in memory would take about 160MB. Furthermore, as the list of word-feature pairs grew larger, searching for a particular word-feature pair in order to increase its count would take an increasingly long time. To make the process more efficient, we implement the word-feature pair counting step as a two part process. First we count the word-feature pairs for a small chunk of the parsed corpus (10 files at a time), and then we combine these counts with a master list of counts using a merge algorithm. A shortened pseudocode description of the algorithm is in Figure 5, and the full Java code is in Appendix D.

The output of this step is a large alphabetical file of word-feature pair counts in the format below, with one word and its set of features per line.

```
word  <feature1>  count1      <feature2>  count2      ...
```

The total count of word-feature pairs, or `|*,*|`, is also recorded in the first line of the file.

```
Function PairCounts(){
  Load parsed_file_list
  WHILE length(parsed_file_list)>0 {
    Initialise counts
    Put names of next 10 files from parsed_file_list into queue
    FOR each file in queue{
      DO{
        Read in next line containing word1, relationship, word2
        Set word1_feature to <word2, relationship, D>
        Set word2_feature to <word1, relationship, H>

        Add 1 to count(word1, word1_feature)
        Add 1 to count(word2, word2_feature)
        Add 2 to total_pair_count

      } UNTIL the end of the file is reached
    }
    Write total_pair_count to tempCountsFile
    FOR each word {
      Output features and counts to tempCountsFile
    }
    Set masterCountsFile to MergeCounts(tempCountsFile, masterCountsFile)
    Delete queue files from parsed_file_list
  }
}

Function MergeCounts(temp, master){
  Open temp
  Open master
  Create outfile
  IF master is empty
    Copy temp to outfile

  ELSE {
    Set combinedCount = totalFeatureCount(temp)+totalFeatureCount(master)
    Write combinedCount to outfile

    WHILE (length(temp)>0 && length(master)>0) {
      IF nextWord(temp) < nextWord(master) {
        Write nextLine(temp) to outfile

        Set temp = rest(temp)
      }
      ELSE IF nextWord(master) < nextWord(temp) {
        Write nextLine(master) to outfile

        Set master = rest(master)
      }
      ELSE IF nextWord(master) == nextWord(temp) {
        Combine feature counts from nextLine(master) and nextLine(temp)
        Write combined line to outfile

        Set temp = rest(temp)
        Set master = rest(master)
```

```
      }
    }
    IF length(master) > 0
      Copy rest of master to outfile
    IF length(temp) > 0
      Copy rest of temp to outfile

    RETURN outfile
 }}
```

***Figure 5.*** *Pseudocode algorithm for counting word-feature pairs.*

At this point we also create a file called the feature index by re-organising the pair counts file. The purpose of the feature index is to speed up later processing by enabling feature counts and word-feature pair counts to be indexed by feature. It contains an alphabetical list of features with their total counts in the parsed corpus, followed by a list of individual words with which the feature co-occurs and their counts. The format is:

```
<feature>   feature_count     word1 count1      word2 count2      ...
```

The third step toward calculating the syntactic feature distributional similarity is to construct a *raw* feature vector for each word by calculating the weight of each feature in its vector. We denote as $F_{raw}(w)$ the set of features that are in this initial feature vector, and it includes all features with which $w$ co-occurs in the corpus. The process is straightforward and follows Equation 3.4, making use of the feature index to quickly reference the count for each feature, $|*,f|$. The Java code is in Appendix E.

The output of the third step is a large alphabetical file of words $w$ followed by the set of features and weights for $f \in F_{raw}(w)$ for which $MI(w,f) > 0$, with the format:

```
w     <f1>  MI(w, f1)   <f2>  MI(w, f2)   ...
```

Once a weighted feature vector has been constructed for each word, we can calculate $LEP_{ws}(u,v)$ for any two words $u$ and $v$ by applying the *sim* function from Equation 3.5 to their vectors. Like the authors of (Geffet & Dagan, 2004) we do not use the raw feature vectors as produced by the previous step. Instead we apply filtering techniques to get rid of rare and low-weight features and produce a vector of *active features*, $F(w)$.

Optimally, a feature's weight relative to the other feature weights within a word's vector should reflect that feature's importance in characterising the word. As pointed out in (Geffet & Dagan, 2004), the function $MI(w,f) = \log_2 \frac{P(w,f)}{P(w)P(f)}$ (Equation 3.4) tends to assign inappropriately high weights to rare features. This affects the calculation of $sim(u,v)$ when $u$ and $v$ share rare features that are not important in characterising those words but nevertheless cause the words to have a high similarity score. To combat this problem we filter out features that occur less than 10 times in the corpus. The feature frequency threshold of 10 was arrived at after informally experimenting with different feature filters ranging from 0 to 20. Coincidentally this is the same threshold that the authors of (Geffet & Dagan, 2004) used for their paper, which makes sense considering that their corpus was of a similar size (18 million tokens versus our 20 million).

We also filter out features that have a weight of less than 4.0. This is because low-weight features within a word's feature vector are not important in characterising the word but add noise to the similarity calculation. As with the feature frequency filter we experimented with several values for the feature weight filter and again arrived at the same threshold that was used in (Geffet & Dagan, 2004).

To calculate $sim(u,v)$ in this implementation we first consult the feature index created earlier to compile a list of active features, ignoring those features with frequency less than 10. Then, when comparing the feature vectors for $u$ and $v$ we only consider features that appear on the active features list and that have a weight greater than 4.0. The pseudocode description of the implementation is in Figure 6.

```
Function LEPws(u,v) {

  Open feature index
  FOR each feature in the feature index {
    IF count(feature) > 10
      Add feature to valid_features
  }
  FOR each f in F_raw(u) {
    IF valid_features contains f {
      IF weight(u,f)>4{
        Add f to F(u)
        Add weight(u,f) to feature_sum(u)
      }
    }
  }

  Set numerator to 0
  FOR each f in F_raw(v) {
    IF valid_features contains f {
      IF weight(v,f)>4 {
        add weight(v,f) to feature_sum(v)
        IF F(u) contains f
          Add (weight(u,f) + weight(v,f)) to numerator
      }
  }

  Set sim = numerator / (feature_sum(u) + feature_sum(v))
  RETURN sim
}
```

***Figure 6.*** *Pseudocode explaining implementation of LEPws(u,v)*

### 3.1.3 Thesaurus-based Similarity

In the paper (Geffet & Dagan, 2004) the authors report that the syntactic feature distributional similarity measure just explained achieves about 54% precision in predicting lexical entailment among the top-10 most similar words that it assigns to a given random noun. The purpose of this project is to see whether replacing the web-based co-occurrence metric $LEP_{co}$ with more accurate LEP metrics in the Bar Ilan model will improve the model's overall accuracy. While 54% precision is better than one might initially expect the $LEP_{co}$ metric to achieve, it is still not terribly accurate. For this reason we consider a third estimate for lexical entailment probability based on a

pre-existing thesaurus which was generated by Dekang Lin, a computational linguistics researcher at the University of Alberta and author of (Lin, 1998)[8].

The Lin thesaurus lists vocabulary words followed by up to 200 of their most similar words and a similarity score between 0 and 1. This makes it very easy for our program to calculate $LEP_{thes}(u,v)$ by simply searching for word $v$ among word $u$'s top 200 most similar word list, and using the similarity score that follows directly as the result for $LEP_{thes}(u,v)$.

## 3.2 Comparison of LEP Metrics

The main objective of this project is to determine whether replacing the co-occurrence $LEP_{co}$ metric from the Bar Ilan model with more accurate estimates for $P(Tr_u = 1 \mid T_v)$ will improve the model's overall accuracy in predicting lexical and textual entailment. Therefore we must first determine whether the proposed $LEP_{ws}$ and $LEP_{thes}$ metrics are indeed more accurate at predicting lexically entailing word pairs than the $LEP_{co}$ estimate.

### 3.2.1 Judgement Criteria

To test each LEP estimate's accuracy we follow the procedure that was used in (Geffet & Dagan, 2004) to test the accuracy of $LEP_{ws}$. For each of the three LEP metrics we randomly pick 20 nouns (which occur more than 50 times in the corpus) and calculate the top-40 highest scoring words for each noun. (Only 10 random nouns were tested for the web-based co-occurrence metric to limit calculation costs; see Chapter 3.1.1 for an explanation.) Then two independent judges manually judge the resulting word pairs for

[8] The thesaurus is available online from (Lin, Downloads).

lexical entailment. Finally we calculate the precision of each *LEP* metric for its top-10, top-20, top-30, and top-40 most similar words.

The judges determine whether lexical entailment holds between word pairs by testing for a relationship termed *meaning entailing substitutability*. This criterion, introduced in (Geffet & Dagan, 2004), identifies whether some context exists in which one word from the pair can be substituted for the other in a sentence while retaining the sentence's meaning. For example, in the sentence *We ate pizza for lunch*, we can substitute the word *food* for *pizza* and retain the sentence's meaning, so the meaning entailing substitutability relationship holds for the word pair *pizza-food*. Meaning entailing substitutability is a rigorous indicator of lexical entailment.

### 3.2.2 Testing $LEP_{co}$

To test our implementation of $LEP_{co}$ we chose a set $N$ of 10 random nouns and a vocabulary $V$. Because of the cost involved with using the Alexa Web Search service it was necessary to carefully limit $V$ by first retrieving the number of hits for all the words which occurred more than six times in our corpus (of which there were 144,412) and then limiting $V$ to only words which had 300,000 or more hits on the web. This reduced the list to 76,912 vocabulary words. For each $u \in N$ and $v \in V$ we calculated $LEP_{co}(u, v)$, the probability that $v$ entailed $u$, and $LEP_{co}(v, u)$, the probability that $u$ entailed $v$. Compiling a list of the top scores with the associated word $v$ for each $u$ produced an initial ranked list of the most likely entailed or entailing words for each $u \in N$.

Two problems with the initial list were immediately apparent. The first problem was that the top 30 or so most similar words for every word $u$ had very high positive $LEP_{co}$ scores. This was a result of the Alexa web search engine using stop words which

returned very few hits. For example, at the time of testing[9] the Alexa search engine treated *at* as a stop word, so the number of hits for the query '*at'* was 10 and the number of hits for '*at AND university'* was 126,692,000. Thus $LEP_{co}$(*university,at*) = 126,692,000/10 = 12,669,200. For this reason we deleted from our entailing words list all words with scores greater than 1.

The second problem which surfaced was that the $LEP_{co}$ metric assigns inappropriately high scores to common words. Consider the word *the,* which appears within most documents on the web. For any other word *u,* the number of hits for '*u AND the'* is going to be very close to the number of hits for '*u*' and $LEP_{co}$(*the,u*) will be very close to 1. To correct this problem we applied the same stop list that we used in the probabilistic lexical entailment model (Chapter 2.4.2) and deleted these stop words from our ranked list of entailing words. After applying these two corrections, we added the top-40 words for each noun in $N$ to our list of word pairs for evaluation.

### 3.2.3 Testing $LEP_{ws}$

To test the implementation of $LEP_{ws}$ we took a set $N$ of 20 random nouns and a vocabulary $V$ and found the top 40 words $v \in V$ as calculated by $LEP_{ws}$ ($u$, $v$). In the interest of limiting processing time we controlled $V$ somewhat by only considering words $v \in V$ that appeared more than six times in the corpus.

We call $R(n)$ the set of *related words* to $n$ and it is defined as the set of words $v \in V: F(n) \cap F(v) \neq \emptyset$. Instead of comparing the vector for each $n \in N$ to the vector for each $v \in V$, one pair of words at a time, we followed an algorithm which enabled us to consider only word pairs $(n, v) : v \in R(n)$. This required the creation of a feature weight index, similar to the feature index created for an earlier step, which listed each feature

[9] At the time of writing, the Alexa web search engine appears to have stopped using stop words. A more recent search for '*at'* returned 99,540,437 hits.

followed by its word set *WS*(*f*) of words for which *f* was an active feature. This had the format:

```
<f>   w1    MI(w1,f)    w2    MI(w2,f)    ...
```

To compile the list of top-ranked similar words for noun *n*, we went through *n*'s vector of active features, *F*(*n*), one at a time. We filtered out inactive features using the feature frequency and weight filters which were explained in Chapter 3.1.2. For each $f \in F(n)$, we pulled *WS*(*f*) from the feature weight index. Then, for each $v \in WS(f)$ such that $v \neq n$ and $MI(v,f) >$ *weight filter*, we added *v* to *R*(*n*) if it wasn't there already and started a running numerator total for *v* to which we added $MI(n,f)+MI(v,f)$. Finally, once we had run through *n*'s entire set of active features, we divided the running numerator total for each $v \in R(n)$ by $\sum_{f \in F(n)} MI(n,f) + \sum_{f \in F(v)} MI(v,f)$.

It should also be noted that in compiling the list of active features *F*(*n*) for each noun *n*, it was advantageous to exclude from *F*(*n*) a list of stop features containing the 40 most common features from the corpus in terms of inverse word frequency. Each of the stop features occurred with over 10,000 different words in the corpus. Including these common features would have made *R*(*n*) extremely large, and because *P*(*f*) was so large for these frequent features, their *MI* weight tended to be small for every word they occurred with. Filtering them out of *F*(*n*) eliminated a lot of unnecessary processing.

A diagram depicting the relationship between *n*, *F*(*n*), and *R*(*n*) is in Figure 7. A solid line in the diagram between a word *w* and a feature *f* indicates that $MI(w,f) > 4.0$. Also, all the features $f \in F(n)$ have frequency greater than 10 and are not part of the stop feature list.

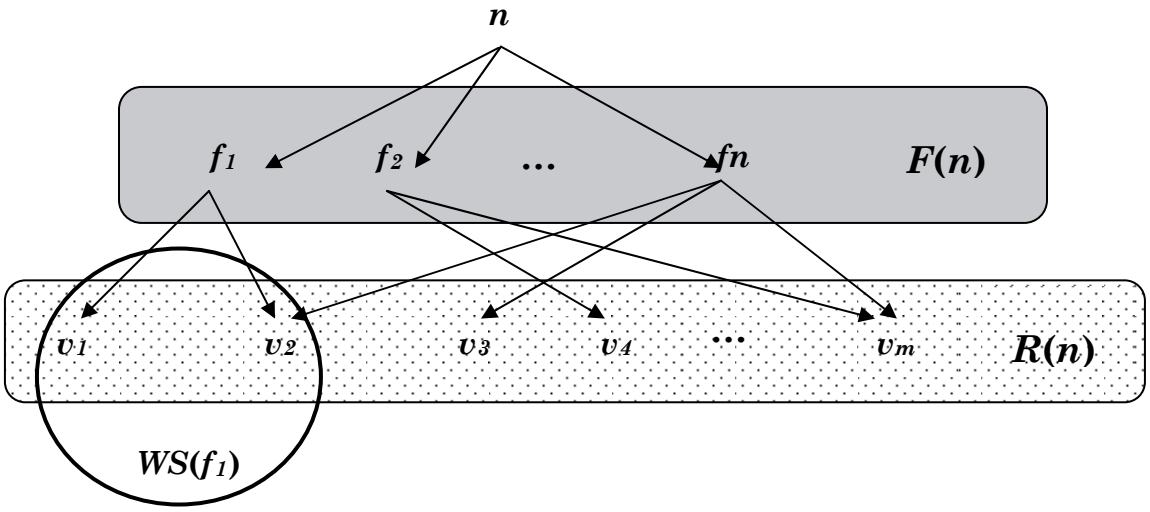


***Figure 7.*** *Diagram depicting the relationship between n, F(n), R(n), and WS(f).*

A pseudocode description of the whole algorithm is below, and the entire Java code is in Appendix F.

```
Function RankLEPws(N){
  FOR each n in N {
    FOR each f in F_raw(n) {
      IF (count(f)>10 && MI(n,f)>4 && f not elem of stop feature list) {
        FOR each v in WS(f) {
          IF MI(v,f) > 4 {
            IF R(n) contains v
              Add MI(n,f) + MI(v,f) to tally(v)
            ELSE
              Add v to R(n)
              Set tally(v) to (MI(n,f) + MI(v,f))
          }
        }
      }
    }
    FOR each v in R(n), score(v)=tally(v)/(feature_sum(n)+feature_sum(v))
    Order R(n) by score(v), descending
    Write top-40 R(n) to output file
  }
}
```

***Figure 8.*** *Pseudocode description of algorithm used to calculate top-40 words for random nouns as evaluated by* $LEP_{ws}$.

### 3.2.4 Testing $LEP_{thes}$

No extra calculations were required to compile the top-40 most similar words for our 20 nouns as judged by $LEP_{thes}$ because the Lin thesaurus already listed these nouns followed by a ranked list of their most similar words. All that was necessary was to extract the appropriate lists from the Lin thesaurus file.

### 3.2.5 Results

Figure 7 lists the top-20 highest scoring words for the noun *policy* as determined by each of the three LEP metrics. Words that do not have a lexical entailment relationship with *policy* are marked with an asterisk.

| | $LEP_{co}$ | | | $LEP_{ws}$ | | | $LEP_{thes}$ | |
|---|---|---|---|---|---|---|---|---|
| 1 | *policyholder | 0.843 | 1 | policies | 0.156 | 1 | strategy | 0.223 |
| 2 | *privacy | 0.819 | 2 | *management | 0.121 | 2 | law | 0.217 |
| 3 | *independent | 0.760 | 3 | strategy | 0.112 | 3 | economic policy | 0.209 |
| 4 | *devious | 0.754 | 4 | *trade | 0.111 | 4 | foreign policy | 0.207 |
| 5 | protectionism | 0.750 | 5 | *issues | 0.107 | 5 | reform | 0.203 |
| 6 | *vaults | 0.750 | 6 | *development | 0.103 | 6 | measure | 0.203 |
| 7 | *us | 0.742 | 7 | *research | 0.099 | 7 | stance | 0.198 |
| 8 | *deviations | 0.725 | 8 | plan | 0.097 | 8 | plan | 0.197 |
| 9 | *identifiable | 0.710 | 9 | *government | 0.097 | 9 | regulation | 0.194 |
| 10 | *disclose | 0.683 | 10 | law | 0.097 | 10 | *principle | 0.186 |
| 11 | *about | 0.675 | 11 | process | 0.097 | 11 | legislation | 0.186 |
| 12 | *constitutes | 0.673 | 12 | *within | 0.097 | 12 | rule | 0.184 |
| 13 | *macroeconomic | 0.667 | 13 | *education | 0.096 | 13 | program | 0.182 |
| 14 | *contact | 0.661 | 14 | *upon | 0.096 | 14 | *proposal | 0.176 |
| 15 | *democracies | 0.650 | 15 | *matters | 0.095 | 15 | *action | 0.170 |
| 16 | *brawl | 0.649 | 16 | *uk | 0.095 | 16 | *guideline | 0.170 |
| 17 | *cloze | 0.646 | 17 | *expenditure | 0.094 | 17 | initiative | 0.168 |
| 18 | *deviant | 0.643 | 18 | *value | 0.092 | 18 | *decision | 0.167 |
| 19 | *insurer | 0.643 | 19 | *resources | 0.092 | 19 | *development | 0.164 |
| 20 | *defamatory | 0.643 | 20 | *sector | 0.091 | 20 | *effort | 0.163 |

***Figure 9.*** *Top 20 most probably entailed or entailing words for the noun* policy *for each LEP metric.*

A quick glance at this table shows that the web-based co-occurrence $LEP_{co}$ metric has the most words without a lexical entailment relationship to *policy* listed among its top-20, while the $LEP_{thes}$ metric has the most actually entailing words in its list. This suggests that the $LEP_{thes}$ metric is more accurate at judging lexical entailment between words than $LEP_{co}$ and $LEP_{ws}$.

The result of manual testing by the two independent judges supports this suggestion. Calculating the top-40 most similar words for the 20 random nouns with the $LEP_{thes}$ and $LEP_{ws}$ metrics and for the 10 random nouns with the $LEP_{co}$ metric produced a total of 1960 independent word pairs. These pairs were randomly split between the two judges and manually judged for the presence of a textual entailment relationship. Each judge also tested 50 pairs from the other's list, so that a total of 100 word pairs were judged by both judges in order to assess their agreement. In the few cases where the two judges disagreed, the author made a judgement as a tiebreaker.

The table below shows the precision values for the top-10, 20, 30, and 40 highest scoring words as judged by each of the annotators.

| | ***LEP*$_{co}$** | | | ***LEP*$_{ws}$** | | | ***LEP*$_{thes}$** | | |
|---|---|---|---|---|---|---|---|---|---|
| | Judge 1 | Judge 2 | Total | Judge 1 | Judge 2 | Total | Judge 1 | Judge 2 | Total |
| Top 10 | .189 | .151 | .180 | .475 | .333 | .405 | .815 | .766 | .786 |
| Top 20 | .167 | .160 | .165 | .361 | .282 | .319 | .795 | .713 | .745 |
| Top 30 | .174 | .179 | .177 | .303 | .263 | .280 | .729 | .645 | .679 |
| Top 40 | .161 | .179 | .168 | .285 | .235 | .257 | .689 | .581 | .631 |

***Figure 10.*** *Precision values for Top-10/20/30/40 words as judged by each of three LEP metrics.*

The results show that $LEP_{thes}$ performed significantly better than $LEP_{co}$ or $LEP_{ws}$ at all Top-*N* levels. It consistently achieved 46 or more percentage points of precision higher than $LEP_{co}$ and 37 or more percentage points higher than $LEP_{ws}$. Both $LEP_{ws}$ and $LEP_{thes}$ produced 800 word pairs, but 530 of the 800 pairs produced by $LEP_{thes}$ were judged a9s entailing versus only 216 of the 800 produced by $LEP_{ws}$.

The judges showed fairly high agreement on their mutually judged pairs. Overall they agreed on 89 of 100 pairs. For the 11 pairs on which they disagreed, Judge 1 believed that 5 were entailing and Judge 2 believed that 6 were entailing. The associated Kappa value is 0.768, indicating 'substantial agreement' (Landis & Koch, 1997). A confusion matrix that shows the distribution of their mutually judged pairs is in Figure 11.

| | | Judge 1 | | |
|---|---|---|---|---|
| | | TRUE | FALSE | Total |
| Judge 2 | TRUE | 33 | 6 | 39 |
| | FALSE | 5 | 56 | 61 |
| | Total | 38 | 62 | 100 |

***Figure 11.*** *Confusion matrix for two judges' assessment of a sample of word pairs.*

### 3.2.4 Converting Similarity Metrics to Lexical Entailment Probabilities

One may notice that the lexical entailment probability scores for $LEP_{ws}$ and $LEP_{thes}$ in Figure 7 are much lower than we would expect the actual probabilities to be. For example, we have $LEP_{thes}(policy, law) = 0.217$, even though one might expect these two words to entail one another in more than 21.7% of contexts. For the purposes of this project, the disparity between the calculated lexical entailment probability and the actual entailment probability is not important. What matters are the relative lexical entailment probabilities of different words – i.e. that entailing word pairs have higher scores than non-entailing word pairs. This is because we judge a sentence pair for lexical entailment based on the relationship of $P(Tr_h = 1|t)$ to $P(Tr_h = 1)$, and we tune $P(Tr_h = 1)$ empirically from the data. So if LEP estimates are uniformly lower than actual lexical entailment probabilities between words, the overall value of $P(Tr_h = 1|t)$ will be low as well. But since we tune $P(Tr_h = 1)$ to fit the data, we should still be able to find a value of $P(Tr_h = 1)$ that separates entailing from non-entailing

sentences as long as entailing word pairs are uniformly ranked higher than non-entailing word pairs.

# Chapter Four: Embedding the New Lexical Entailment Probability Metrics

In this section we train the probabilistic lexical entailment model on the RTE-1 development dataset and test it on the RTE-1 test dataset, embedding each of the three LEP metrics in turn. Analysis of the results shows that while replacing the web-based co-occurrence LEP metric from the Bar Ilan model with the more accurate syntactic feature distributional similarity and thesaurus based metrics does not improve the model's ability to predict textual entailment, it does cause the model to perform better in the lexical entailment recognition task. This chapter describes and presents the results of our experiments.

## 4.1 Smoothing

One of the main differences between the $LEP_{co}$ metric and the other two is that the $LEP_{co}$ metric is capable of calculating a similarity score for any pair of words, while the $LEP_{ws}$ estimate can only calculate similarities for words that share syntactic features and the $LEP_{thes}$ metric can only calculate similarities for word pairs listed within the thesaurus. This affects the textual entailment model's alignment when the model encounters a hypothesis word $h$ that is not comparable to any of the text words. We

cannot just say that the lexical entailment probability for such words is 0 because the overall entailment probability for the sentence is the product of the maximum LEP values for each word, and an LEP of 0 for one word would bring down the probability of the entire sentence. Therefore it is necessary to incorporate smoothing into the model to come up with an LEP score for hypothesis words that have no comparable words from the text. This is similar to using smoothing in bigram models to calculate probabilities for unseen bigrams.

For this project we adopt a simple smoothing method, based on the additive smoothing technique for bigram modelling, which assumes that each unseen bigram occurs once (Chen & Goodman, 1996). This is generally not a strong smoothing method but we use it nonetheless for the sake of simplicity and because it aligns with our probabilistic setting, which says that the probability of generating some hypothesis $h$ totally unrelated to the text is some prior $P(h)$.

In this case we assume that the LEP for incomparable hypothesis words is some small constant δ. In testing I experimented with several values of δ for both the $LEP_{thes}$ and $LEP_{ws}$ metrics. The reason for testing with several values is that there must be a large enough difference between possibly-entailing words (i.e. words for which a LEP can be calculated) and completely incomparable words to reflect the actual fact that the lexical entailment probability for incomparable words is very low. On the other hand we cannot make the LEP for incomparable words so low that one incomparable hypothesis word destroys the entailment probability of an entire sentence. So in this implementation we substituted the values $\delta = \{0.001, 0.01, 0.02\}$ to test across a small range of orders of magnitude.

## 4.2 Textual Entailment Recognition Accuracy

The RTE-1 development and test datasets described in Chapter 2.4.1 are manually annotated for the textual entailment relationship. This makes it straightforward to test the Bar Ilan model's accuracy in predicting textual entailment by first tuning the cut-off

value λ based on the annotated values and calculated entailment probability scores from the development dataset. We then use the empirically tuned value of λ to classify each sentence pair in the test dataset based on its calculated entailment probability. Finally we check the classification of the test data against the manually annotated entailment relationship values to determine the model's overall accuracy.

After implementing the smoothing measure, I ran the textual entailment model on the RTE-1 development dataset once using $LEP_{co}$ and three times each for $LEP_{thes}$ and $LEP_{ws}$, substituting a different smoothing value δ (0.001, 0.01, and 0.02) in each trial. After each run of the model on the development dataset, I fed the resulting scores and their pre-judged entailment values for all sentence pairs to the Weka software which empirically tuned λ using the C4.5 algorithm. The following table displays the resulting values for λ and the accuracy of each model on the training data. It shows that the $LEP_{ws}$ metric was the best at classifying the training data in terms of percent of correctly classified sentences by a very small margin, and that all models had a high occurrence of false positives.

| Metric | δ | λ | Percent Correct | True Positives | False Positives | True Negatives | False Negatives |
|---|---|---|---|---|---|---|---|
| $LEP_{co}$ | | .000517 | 55.9083 | 261 | 228 | 56 | 22 |
| $LEP_{ws}$ | 0.001 | .000018 | 57.3192 | 202 | 161 | 123 | 81 |
| | 0.01 | .000075 | 57.50 | 219 | 176 | 108 | 64 |
| | 0.02 | .000192 | 56.875 | 215 | 173 | 111 | 68 |
| $LEP_{thes}$ | 0.001 | 7.73E-11 | 56.9665 | 264 | 225 | 59 | 19 |
| | 0.01 | .00008 | 56.9665 | 222 | 183 | 101 | 61 |
| | 0.02 | .00016 | 56.9665 | 222 | 183 | 101 | 61 |

***Figure 12.*** *Calculated cut-off values and performance of the Bar Ilan model on the textual entailment recognition task, using the RTE-1 development dataset (567 sentence pairs).*

I then tested each model's performance on the test data by classifying each sentence pair within the test dataset based on the model's associated cut-off value λ. The model read in the calculated entailment probabilities for each sentence pair in the test dataset, and classified sentence pairs for which $P(Tr_h=1 \mid t) \leq \lambda$ as *FALSE* and the rest as *TRUE*. The results are shown in the table on the next page.

| Metric | Smoothing Value (δ) | Correctly Classified | Incorrectly Classified | True Positives | False Positives | True Negatives | False Negatives | Breakdown by Class | TP Rate | FP Rate | Precision | Recall | F-Measure |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***LEP**$_{co}$* | | 457 (57.125%) | 343 (42.875%) | 370 | 313 | 87 | 30 | TRUE | .925 | .783 | .542 | .925 | .683 |
| | | | | | | | | FALSE | .218 | .075 | .744 | .218 | .337 |
| ***LEP**$_{ws}$* | 0.001 | 453 (56.625%) | 347 (43.375%) | 294 | 241 | 159 | 106 | TRUE | .735 | .603 | .550 | .735 | .629 |
| | | | | | | | | FALSE | .398 | .265 | .600 | .398 | .478 |
| | 0.01 | 460 (57.50%) | 340 (42.5%) | 317 | 257 | 83 | 143 | TRUE | .793 | .643 | .552 | .793 | .651 |
| | | | | | | | | FALSE | .358 | .208 | .633 | .358 | .457 |
| | 0.02 | 455 (56.875%) | 345 (43.125%) | 311 | 256 | 144 | 89 | TRUE | .778 | .640 | .549 | .778 | .643 |
| | | | | | | | | FALSE | .360 | .223 | .618 | .360 | .455 |
| ***LEP**$_{thes}$* | 0.001 | 455 (56.875%) | 345 (43.125%) | 369 | 314 | 86 | 31 | TRUE | .923 | .785 | .54 | .923 | .681 |
| | | | | | | | | FALSE | .215 | .078 | .735 | .215 | .333 |
| | 0.01 | 450 (56.25%) | 350 (43.75%) | 307 | 257 | 143 | 93 | TRUE | .768 | .643 | .544 | .768 | .637 |
| | | | | | | | | FALSE | .358 | .233 | .606 | .358 | .450 |
| | 0.02 | 450 (56.25%) | 350 (43.75%) | 307 | 257 | 143 | 93 | TRUE | .768 | .643 | .544 | .768 | .637 |
| | | | | | | | | FALSE | .358 | .233 | .606 | .358 | .450 |

***Figure 13.*** *Performance of the Bar Ilan model on the textual entailment recognition task; RTE-1 test dataset (800 sentence pairs).*

Looking at the results table, the first thing that is noticeable is that the more accurate LEP metrics did not produce more accurate textual entailment models overall. The 'least accurate' LEP metric, $LEP_{co}$, did better than the 'most accurate' metric, $LEP_{thes}$, and nearly as well as the best $LEP_{ws}$ model in terms of the percentage of sentences it classified correctly as textually entailing or non-entailing. Furthermore, the $LEP_{ws}$ and $LEP_{thes}$ models were just about even in terms of correctly classifying sentence pairs, even though $LEP_{thes}$ was more accurate than $LEP_{ws}$ at picking entailing word pairs as discussed in Chapter 3.2.5. By a slim margin, the best of the seven models was the one that used $LEP_{ws}$ with $\delta$=0.01. It correctly classified 57.5% of sentence pairs, with a recall of .793 and precision of .552 on the *TRUE* pairs and recall of .358 with precision of .633 on the *FALSE* sentence pairs.

The second thing that jumps out is that the vast majority of errors for all models were false positives. Every model was overwhelmingly biased toward classifying sentences as entailing. This can be explained by looking at the data visualisation output from the Weka software. Figure 14 shows a plot of similarity scores, with sentence pairs classified as *FALSE* in red and *TRUE* pairs in shown in blue. The data is scattered for better visibility.

The figure shows that there is no apparent distinction in similarity score for *TRUE* and *FALSE* sentence pairs. One would expect that a system which effectively separated *TRUE* and *FALSE* sentences by similarity score would show a large cluster of blue data points toward the top right of the figure, and a cluster of red points in the bottom left. As it stands there is no obvious cut-off point; red and blue points are mixed together for all score values. This is why the models could not set a cut-off value $\lambda$ that effectively separated *TRUE* from FALSE sentence pairs.

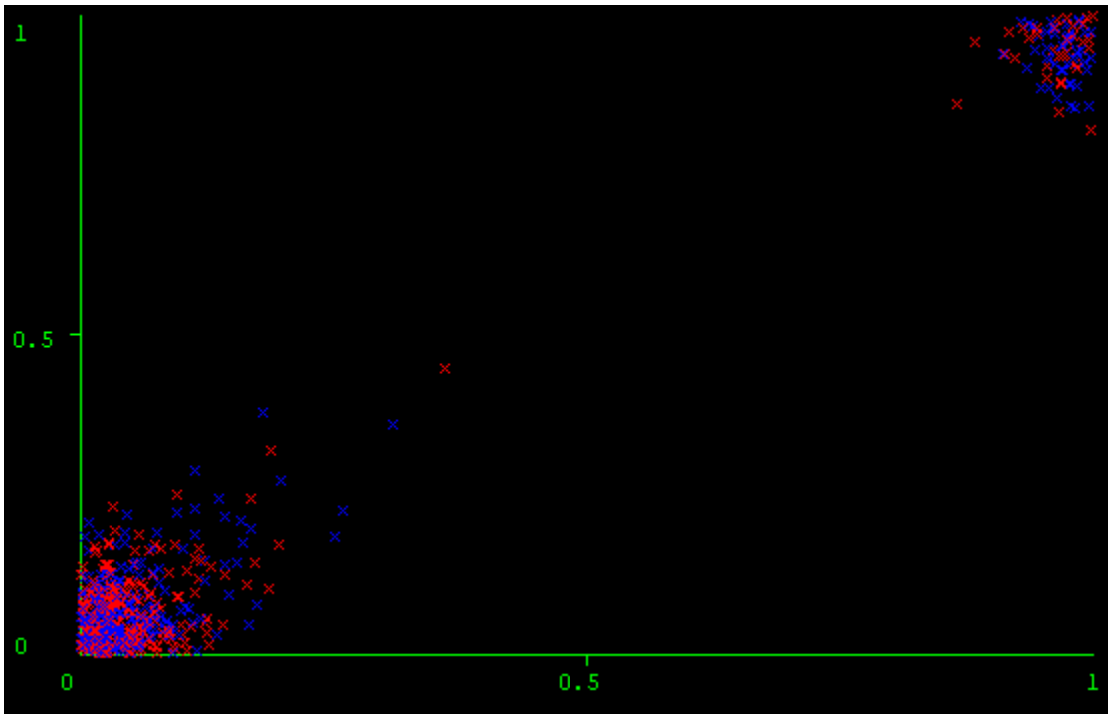


***Figure 14. Textual Entailment Similarity Score v Similarity Score.*** *The visualisation in this figure displays the distribution of similarity scores for sentence pairs classified as TRUE (blue) and FALSE (red). Data has been scattered for better visibility.*

The probabilistic lexical entailment model for recognising textual entailment uses its estimation of the lexical entailment relationship directly as its estimate for the existence of a textual entailment relationship. As was discussed in Chapter 2.3, any model that uses lexical entailment as the sole criterion for estimating the textual entailment relationship between a pair of sentences can only hope to estimate correctly about 69% of the time. However, our best model was only 57.5% accurate, and models that incorporated fairly accurate lexical entailment probability estimates did not perform better than those with inaccurate LEP estimates. The first step toward trying to explain these results is to see whether or not implementing more accurate lexical entailment probability metrics actually improved the model's accuracy on the lexical entailment recognition subtask.

## 4.3 Lexical Entailment Recognition Accuracy

One possible reason why embedding more accurate LEP metrics did not produce more accurate textual entailment systems overall could be that the more accurate LEP estimates did not improve the model's performance on the lexical entailment recognition subtask either. We therefore conduct a second round of tests to see whether this is the case, or whether we can rule it out.

There was no pre-annotated dataset for the lexical entailment relationship so it was necessary to create one. To do this I took a portion of the RTE-1 dataset containing 200 sentence pairs and manually judged each for the lexical entailment relationship. To perform the manual annotation I ignored stop words (from Appendix A) in the text and hypothesis, and for each word $h$ in the hypothesis, I checked to see whether $h$ was lexically entailed by a single word from the text. If all the hypothesis words were lexically entailed, then the sentence was lexically entailed and the sentence pair given the value *TRUE*. If even one hypothesis was not lexically entailed by a word from the text, the entire sentence pair was given the value *FALSE* for the lexical entailment relationship. When judging word pairs for lexical entailment I used the meaning entailing substitutability criterion discussed in Chapter 3.2.1. I also judged any word pair that consisted of different forms of the same word (i.e. *liked, likes*; *ran, running*; *tree, trees*; *Iran, Iranian*) as entailing.

With the annotated lexical entailment dataset I tuned cut-off values for each of the 7 entailment models, including all combinations of LEP metric and smoothing constant $\delta$, using the Weka software. This time I used the same entailment probability scores calculated by each of the models as before but instead of using the textual entailment values I used the new lexical entailment values for classification. Since I did not have separate development and test sets, it was necessary to do a 10-fold cross-validation to come up with accuracy measurements for each model. Also, when tuning the cut-off values, the decision trees created by the C4.5 algorithm had more than one node for

| Metric | Smoothing Value (δ) | Cut off value λ | Correctly Classified | Incorrectly Classified | True Positives | False Positives | True Negatives | False Negatives | Breakdown by Class | TP Rate | FP Rate | Precision | Recall | F-Measure |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***LEP**$_{co}$* | | .03735 | 128 (64%) | 72 (36%) | 79 | 40 | 49 | 32 | TRUE | .712 | .449 | .664 | .712 | .687 |
| | | | | | | | | | FALSE | .551 | .288 | .605 | .551 | .576 |
| ***LEP**$_{ws}$* | 0.001 | .00590 | 144 (72%) | 56 (28%) | 75 | 20 | 69 | 36 | TRUE | .676 | .225 | .789 | .676 | .728 |
| | | | | | | | | | FALSE | .775 | .324 | .657 | .775 | .711 |
| | 0.01 | 5.865E-4 | 145 (72.5%) | 55 (27.5%) | 77 | 21 | 68 | 34 | TRUE | .694 | .236 | .786 | .694 | .737 |
| | | | | | | | | | FALSE | .764 | .306 | .667 | .764 | .712 |
| | 0.02 | .001714 | 145 (72.5%) | 55 (27.5%) | 77 | 21 | 68 | 34 | TRUE | .694 | .236 | .786 | .694 | .737 |
| | | | | | | | | | FALSE | .764 | .306 | .667 | .764 | .712 |
| ***LEP**$_{thes}$* | 0.001 | 7.468E-5 | 159 (79.5%) | 41 (20.5%) | 90 | 20 | 69 | 21 | TRUE | .811 | .225 | .818 | .811 | .814 |
| | | | | | | | | | FALSE | .775 | .189 | .767 | .775 | .771 |
| | 0.01 | 7.468E-4 | 148 (74%) | 52 (26%) | 87 | 28 | 61 | 24 | TRUE | .784 | .315 | .757 | .784 | .770 |
| | | | | | | | | | FALSE | .685 | .216 | .718 | .685 | .701 |
| | 0.02 | .001494 | 153 (76.5%) | 47 (23.5%) | 86 | 22 | 67 | 25 | TRUE | .775 | .247 | .796 | .775 | .785 |
| | | | | | | | | | FALSE | .753 | .225 | .728 | .753 | .740 |

***Figure 15.*** *Performance of the Bar Ilan model on the lexical entailment recognition task, using the author-annotated dataset of 200 sentence pairs, with testing done using 10-fold cross validation.*

most models with this dataset. I used the DecisionStump algorithm instead. The DecisionStump is basically a decision tree limited to one node. The results are shown in the table in Figure 15.

As the table shows, the Bar Ilan model correctly classified more sentence pairs for the lexical entailment recognition task than it did for recognising textual entailment in every case. Furthermore, embedding more accurate LEP metrics corresponded with overall higher accuracy on the lexical entailment task. Recall that the $LEP_{co}$ metric achieved a precision of only 18% for its top ten highest scoring word pairs in the tests in Chapter 3.2.5, while the $LEP_{ws}$ metric achieved 40.5% precision and the $LEP_{thes}$ metric did the best with 78.6%. When embedded within the Bar Ilan model, these metrics produced 64%, 72.5%, and 79.5% correctly classified sentence pairs respectively on the lexical entailment recognition task. This shows that models with more accurate metrics for recognising entailing word pairs within the alignment model produced better overall lexical entailment recognition at the sentence level.

The numbers of false positives and false negatives were more balanced for every model than they were in the textual entailment recognition task, where false positives were overwhelmingly more common. For the $LEP_{co}$ and $LEP_{thes}$ metrics the number of false positives was slightly higher than the number of false negatives, while for the $LEP_{ws}$ metric it was the other way around. Overall the strongest model was the one that used $LEP_{thes}$ with $\delta$=0.001. It classified 79.5% of the sentence pairs correctly, with recall of .811 and precision of .818 for the TRUE pairs and recall of .775 with precision of .767 on the FALSE sentence pairs.

The reason why the improved LEP metrics led to improved lexical entailment recognition can be seen by examining the word alignment produced on a sentence pair by each of the three metrics. Figure 16 shows an example sentence pair from the RTE-1 dataset and the various alignments produced by each LEP metric.

*Text:* It asserted that within this framework, we draw your attention (People's Congress members) to Legislation 24 dealing with foreign currency circulation, which is no longer applicable and it has become one of the most significant obstacles to economic and investment activities.

*Hypothesis:* Article 24 is obsolete, and is hindering the economy.

*Textual Entailment:* TRUE

*Lexical Entailment:* FALSE

| **Hypothesis Word** | ***LEP*$_{co}$ Alignment** | ***LEP*$_{ws}$ Alignment** | ***LEP*$_{thes}$ Alignment** | **Manual Alignment** |
|---|---|---|---|---|
| article | no | legislation | legislation | legislation |
| 00 | 00 | 00 | 00 | 00 |
| obsolete | no | (none) | (none) | (none) |
| hindering | no | (none) | (none) | (none) |
| economy | no | investment | currency | economic |

***Figure 16.*** *Alignment for sentence pair (ID=1251) produced by each LEP metric.*

This is a typical alignment by each of the three LEP metrics illustrating some common mistakes. The most inaccurate *LEP*$_{co}$ metric aligns every hypothesis word not stated verbatim in the text with the text word *no* because *no* is a very common word and, as explained in Chapter 3.2.2, common words tend to produce inappropriately high values for *LEP*$_{co}$. Thus its prediction of lexical entailment for this sentence pair is inaccurate. *LEP*$_{ws}$ and *LEP*$_{thes}$ both do fairly well on recognising that neither *obsolete* nor *hindering* is lexically entailed by any single word from the text, but they also both make a typical mistake on the term *economy* by aligning it with a related, but not technically entailing, word. Both are more accurate than *LEP*$_{co,}$ but not perfect. Another example sentence pair follows in Figure 17.

In this case, the most accurate LEP metric, *LEP*$_{thes}$, produced the most correct alignments, while *LEP*$_{co}$ and *LEP*$_{ws}$ produced the least. Most of the incorrect alignments produced by *LEP*$_{co}$ made no sense (i.e. *speculated* → *occurrence*; *tried* → *pointing*; *use* → *affirmed*). The *LEP*$_{ws}$ metric aligned just as many words incorrectly, but the words it misaligned tended to be related in some way (i.e. For *speculated* → *affirmed* both words are verbs in the past tense having to do with making a statement). However it was unable to pick up that *id* was short for *identity,* which may have been

solved had the training corpus been larger. The $LEP_{thes}$ metric made the least mistakes and produced

*Text:* On the other hand, Hilal affirmed that some voters were abroad or did not obtain their election cards, pointing to the occurrence of rare violations such as, for example, the desire of some voters to vote by using their identity cards.

*Hypothesis:* Hilal speculated that some of voters had not received voting passes, or had tried to use their id cards.

*Textual Entailment:* TRUE

*Lexical Entailment*: FALSE

| **Hypothesis Word** | $LEP_{co}$ **Alignment** | $LEP_{ws}$ **Alignment** | $LEP_{thes}$ **Alignment** | **Manual Alignment** |
|---|---|---|---|---|
| hilal | hilal | hilal | hilal | hilal |
| speculated | occurrence | affirmed | (none) | (none) |
| some | some | some | some | some |
| voters | voters | voters | voters | voters |
| received | affirmed | using | (none) | obtain |
| voting | voters | pointing | vote | vote |
| passes | voters | (none) | (none) | cards |
| tried | pointing | using | abroad | (none) |
| use | affirmed | such | obtain | using |
| id | pointing | (none) | identity | identity |
| cards | cards | cards | cards | cards |

***Figure 17.*** *Alignment for sentence pair (ID=1250) produced by each* LEP *metric.*

the most accurate alignments. Overall, the more accurate the LEP metric, the better alignments the model produces and the more accurate the model is at predicting lexical entailment.

There is still room for improvement on the lexical entailment recognition task. The two errors that the model made were false positives – marking *FALSE* sentences as lexically entailing – and false negatives – marking *TRUE* sentences as not entailing. Examining some of the incorrectly classified sentence pairs shows the reason for the errors. The false negatives were caused when the LEP metric failed to recognise entailing word pairs, and in most cases the missed word pairs were different forms of the same word. For instance, in the example from Figure 18 which shows an alignment created by the $LEP_{thes}$ metric, the hypothesis words are all actually entailed by a different form of the

same word (*study→studies, hibernation→hibernating, mammal→mammals, german→germany*). However, the $LEP_{thes}$ fails to recognise most of these entailments and produces a very low score for the sentence pair, resulting in its classification of not entailing. In order to correct this deficiency we may want to incorporate stemming or some other method of recognising different variations of the same word into our LEP metrics.

*Text:* The German-based team say their study is the only report of prolonged hibernation in a tropical mammal.

*Hypothesis:* The team studies hibernating mammals in Germany.

*Lexical Entailment*: TRUE

| **Hypothesis Word** | ***LEP*$_{thes}$ Alignment** | **Manual Alignment** |
|---|---|---|
| team | team | team |
| studies | (none) | study |
| hibernating | (none) | hibernation |
| mammals | (none) | mammal |
| germany | german | german |

***Figure 18.*** *Alignment for sentence pair (ID=211) produced the $LEP_{thes}$ metric and manual alignment.*

The other errors that the model made were false positives. Some of the false positives happened when the model simply aligned word pairs that were actually not entailing. Others were caused by an inconsistency in the model's estimation of the cut-off value, $\lambda = P(Tr_h=1)$. Examining the table in Figure 15, one will notice that in every case where the model uses smoothing, the cut-off value is less than the smoothing value. This means that *FALSE* sentences which are correctly identified as having just one un-entailed hypothesis word will still be classified as lexically entailing. This is an inconsistency that can only be fixed by adjusting the probabilistic model to estimate $P(Tr_h=1)$ in a more logical way than simple empirical tuning.

## 4.4 Discussion

By testing the performance of the several Bar Ilan models on the textual and lexical entailment recognition tasks, we find that embedding more accurate LEP estimates

does in fact increase the model's overall accuracy in predicting lexical entailment but does not affect the model's accuracy in the textual entailment recognition task. The figure below compares the seven models that were tested in terms of the percentage of sentence pairs they classified correctly. Looking at this table one can see that the accuracy of the seven models does not vary significantly for the textual entailment task. For the lexical entailment task, though, the model using the least accurate $LEP_{co}$ metric classifies only 64% of sentence pairs correctly, while the model using the most accurate $LEP_{thes}$ metric classifies 79.5% of pairs correctly when used with δ=0.001.

| **LEP Metric** | ***LEP*$_{co}$** | ***LEP*$_{ws}$** | | | ***LEP*$_{thes}$** | | |
|---|---|---|---|---|---|---|---|
| **Precision of Top-10 similar words** | .180 | .405 | | | .786 | | |
| **δ** | | 0.001 | 0.01 | 0.02 | 0.001 | 0.01 | 0.02 |
| **Textual Entailment % Correctly Classified** | 57.125 | 56.625 | 57.5 | 56.875 | 56.875 | 56.25 | 56.25 |
| **Lexical Entailment % Correctly Classified** | 64 | 72 | 72.5 | 72.5 | 79.5 | 74 | 76.5 |

***Figure 19.*** *Percentage of sentence pairs correctly classified by model. The precision scores are taken from Figure 10.*

Another way to compare the models' performance on the two tasks is by adapting a common evaluation measure from Information Retrieval called the F-measure (van Rijsbergen, 1979). The F-measure is calculated using the formula:

$$F(r,p) = \frac{2rp}{r+p} \tag{4.1}$$

where $r$ denotes recall and $p$ denotes precision of an information retrieval system. As explained in (Rennie, 2004), the F-measure is simply the harmonic mean of the recall and precision values. That paper defines the harmonic mean $H$ of the set of numbers $X = \{x_1 \ldots x_n\}$ as:

$$\frac{1}{H} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{x_i}$$

(4.2)

Our systems have two sets of recall and precision values – one for the sentence pairs classified as *TRUE* and another for the pairs classified as *FALSE*. We can adapt the F-measure to our system by simply taking the harmonic mean of all four numbers, substituting $X=\{r_T,p_T,r_F,p_F\}$ in Equation 4.2. The figure below shows the harmonic mean of each model on the textual and lexical entailment recognition tasks.

| **LEP Metric** | $LEP_{co}$ | $LEP_{ws}$ | | | $LEP_{thes}$ | | |
|---|---|---|---|---|---|---|---|
| **Precision of Top-10 similar words** | .180 | .405 | | | .786 | | |
| **δ** | N/A | 0.001 | 0.01 | 0.02 | 0.001 | 0.01 | 0.02 |
| **Textual Entailment Harmonic Mean** | 0.451603 | 0.543629 | 0.537223 | 0.5331385 | 0.4470781 | 0.5274415 | 0.527441 |
| **Lexical Entailment Harmonic Mean** | 0.627127 | 0.71954 | 0.724462 | 0.7244621 | 0.7921352 | 0.7340627 | 0.762159 |

***Figure 20.*** *Harmonic mean by model.*

Using the harmonic mean as a judge, the two worst-performing models for the textual entailment recognition task were the $LEP_{co}$ model and the $LEP_{thes}$ model with δ=0.001. Referring back to Figure 13, we see that both of these models classified 683 of 800 sentence pairs as textually entailing, leading to a very high number of false positives and consequently very low recall for the *FALSE* pairs, which brought down their harmonic means. Overall, the precision of the LEP metric does not seem to have much bearing on the harmonic mean for the textual entailment recognition task. For the lexical entailment task, however, we notice that the most accurate $LEP_{thes}$ metric produces the best harmonic means overall, while the least accurate $LEP_{co}$ metric produces the lowest harmonic mean. Again, high accuracy for the embedded LEP metric correlates to better accuracy on the lexical entailment recognition task but not on the textual entailment recognition task.

It is relatively straightforward to see why the Bar Ilan model performed so much better on the lexical entailment recognition task overall than it did on predicting textual entailment. Recall that the Bar Ilan model is designed to predict lexical entailment directly, and then to use this as the estimate for the textual entailment relationship. It is more closely related to the lexical than the textual entailment prediction task and thus should perform it better.

If we look at the distribution of scores for the lexical entailment dataset next to the distribution of scores from the textual entailment dataset (repeated from Figure 14), we see that while the textual entailment dataset does not obviously separate into clusters of *TRUE* and *FALSE* sentence pairs, the lexical entailment graph shows a much more distinct separation between the sentence pairs classified as *TRUE*, in blue, and the *FALSE* sentence pairs in red. Thus it was possible for the model to pick a cut-off point that better separated the data. However the area where the two clusters meet is still fairly mixed, which explains why the model was still not completely accurate at predicting lexical entailment.

The question that remains is why the gains in lexical entailment recognition accuracy resulting from the embedding of more accurate LEP metrics did not translate to increased accuracy on the textual entailment recognition task. After examining disagreements in sentence pair classification between the lexical entailment and textual entailment datasets, it is apparent that this may be due in part to the fact that lexical entailment recognition is only part of the process of recognising textual entailment; a perfect textual entailment recognition system would need to incorporate other tests in addition.

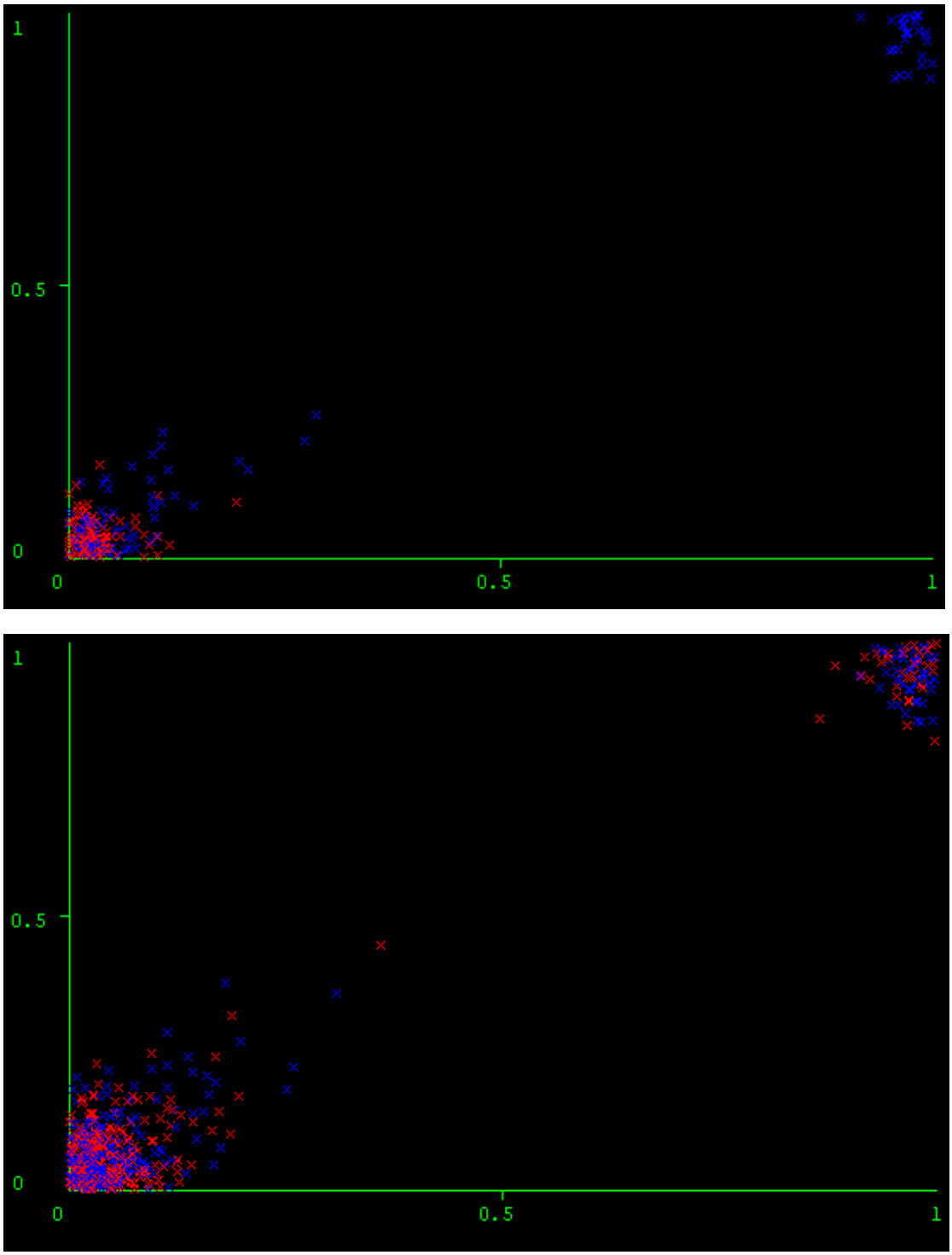


***Figure 21.*** *Distribution of similarity scores for lexical (top) and textual (bottom) entailment datasets. Note the more distinct separation between TRUE (blue) and FALSE (red) sentence pairs for the lexical entailment dataset than for the textual entailment recognition dataset.*

The confusion matrix below compares the lexical and textual entailment classifications for the 200 sentence pairs in the lexical entailment dataset. Of the 200 sentence pairs, 134 (67%) agreed in their lexical and textual entailment classifications. This is close to the value of 69% agreement between lexical and textual entailment classifications estimated in (Glickman, 2006).

| | | Textual Entailment Classification | | |
|---|---|---|---|---|
| | | TRUE | FALSE | Total |
| Lexical Entailment Classification | TRUE | 74 | 37 | 111 |
| | FALSE | 29 | 60 | 89 |
| | Total | 103 | 97 | 200 |

***Figure 22.*** *Comparison of lexical and textual entailment datasets.*

Of the 66 sentence pairs for which the lexical and textual classification was different, 37 (56%) were lexically entailed but not textually entailed. An example of one of these sentence pairs is below:

*Text*: Gunmen loyal to Bosnian Serb nationalist leader Radovan Karadzic conquered 70% of Bosnia with their arsenal of tanks, aircraft and howitzers bequeathed by the Yugoslav army.

*Hypothesis*: Radovan Karadzic is the leader of Bosnia.

***Figure 23.*** *Example of sentence pair that is lexically but not textually entailed.*

In this hypothesis there are four words that are not stop words and all of them are stated verbatim in the text, so the lexical entailment relationship holds for the sentence pair. But the underlying meaning of the text does not entail the underlying meaning of the hypothesis, so it is not textually entailed. Our probabilistic lexical entailment model is incapable of picking up this difference. In order to classify this sentence directly, we would need a textual entailment recognition model that incorporated some type of semantic analysis.

The other 29 sentence pairs for which the lexical and textual entailment classifications did not match up were textually entailed but not lexically entailed. A few examples of such pairs are repeated below to illustrate some of the issues that cause these errors.

| | |
|---|---|
| Pair 1 | |
| *Text*: | Treasures belonging to Hollywood legend Katharine Hepburn have raised £3.2m at a two-day auction in America. |
| *Hypothesis*: | A two-day auction of property belonging to actress Katharine Hepburn brought in 3.2 million pounds. |
| Pair 2 | |
| *Text*: | The memorandum noted the United Nations estimated that 2.5 million to 3.5 million people died of AIDS last year. |
| *Hypothesis*: | Over 2 million people died of AIDS last year. |
| Pair 3 | |
| *Text*: | Jakarta lies on a low, flat alluvial plain with historically extensive swampy areas; the parts of the city farther inland are slightly higher. |
| *Hypothesis*: | The parts of Jakarta away from the coast are on slightly higher land. |
| Pair 4 | |
| *Text*: | The country's largest private employer, Wal-Mart Stores Inc., is being sued by a number of its female employees who claim they were kept out of jobs in management because they are women. |
| *Hypothesis*: | Wal-Mart sued for sexual discrimination. |
| Pair 5 | |
| *Text*: | Government forces killed the head of the Armed Islamic Group, or GIA, which has claimed responsibility for killing 61 foreigners in the last year. |
| *Hypothesis*: | The abbreviation GIA stands for Armed Islamic Group. |

***Figure 24.*** *Examples of sentence pairs which are textually but not lexically entailed.*

The problems with the first three pairs can would be relatively straightforward to manage. The issue with Pair 1 is that the lexical entailment model does not recognise that *£3.2m* is an abbreviation for *3.2 million pounds*; this could be solved by including *£* and *m* as words of the text and training the model to recognize that *£* entails *pounds* and *m* entails *million*. The issue for Pair 2 is that our model does not have any mechanism to do numerical analysis; it converts all digits to 0 and cannot recognise that *2.5 million to 3.5 million* entails *over 2 million*. This might be solved by refraining from converting digits to 0 and incorporating some sort of numerical analysis, perhaps by interpreting words like *over, under,* and *approximately* in their mathematical senses of >, <, and ≈. The reason why Pair 3 is not lexically entailed is that our model does not consider phrase entailment; if it did, it would recognise that *farther inland* entails *away*

*from the coast*. It may be possible to implement phrasal entailment by applying techniques developed for statistical phrase-based translation (Koehn, Och, & Marcu, 2003) or paraphrase recognition (Quirk, Brockett, & Dolan, 2004) (Dolan, Quirk, & Brockett, 2004). Pair 4 and Pair 5 have deeper problems. For both sentence pairs it is only possible to conclude that the textual entailment relationship holds by combining semantic understanding and world knowledge, which are difficult research areas in natural language understanding.

As the errors above illustrate, recognising lexical entailment is only one of the requirements for recognising textual entailment. This is why the gains in lexical entailment recognition which resulted from embedding more accurate LEP estimates within the model did not translate to gains in textual entailment recognition.

# Chapter Five: Conclusion

The overall aim of this project was to see whether it was possible to improve the accuracy of the Bar Ilan probabilistic lexical entailment model on the textual entailment recognition task by replacing its simple web-based co-occurrence lexical entailment probability estimate with more accurate LEP metrics. We implemented the Bar Ilan $LEP_{co}$ metric as well as two other estimates based on the distributional similarity of syntactic features ($LEP_{ws}$) and an existing thesaurus ($LEP_{thes}$). We then tested the accuracy of each metric to verify that the two latter metrics were more precise than the first. Finally we embedded each LEP metric within the Bar Ilan model to test its effect on the model's performance in the lexical and textual entailment recognition tasks.

The results of these tests showed that increasing the LEP metric's accuracy corresponded with an increase in the model's lexical entailment prediction accuracy from 64% to 79.5%, but had no major effect on the model's textual entailment recognition accuracy which hovered around 56-57%.

An analysis of the results shows that further gains in lexical entailment recognition could be made in at least two ways. First, to enable the model to recognise when a hypothesis word is entailed by another form of itself within the text, it may be beneficial to apply stemming to the words of the text and hypothesis before calculating the lexical

entailment probability between each pair of words. Second, instead of simply estimating the 'cut-off value,' or prior probability that the hypothesis sentence is true, by empirically tuning it to the data, it may be beneficial to estimate $P(Tr_h=1)$ based on the content of the hypothesis. One way to do this could be to vary $P(Tr_h=1)$ with the length of the hypothesis, since the entailment probability $P(Tr_h=1 \mid t)$ is estimated as the product of the individual lexical entailment probabilities $P(Tr_u=1 \mid v)$ for each $u \in h$, and therefore $P(Tr_h=1 \mid t) \propto c^h$ for some constant $c$. Another possibility might be to base $P(Tr_h=1)$ on the frequency of each $u \in h$ within some corpus. All of these modifications to the lexical entailment model might be interesting areas for future research.

However, the results of our tests also suggest that improving the Bar Ilan model's lexical entailment recognition ability will not lead to major gains in its textual entailment recognition ability because lexical entailment recognition is only a part of what a system would need to predict textual entailment accurately. Other requirements of a textual entailment recognition system might include semantic analysis, numerical analysis, and phrase-based lexical entailment recognition. Further research into the textual entailment recognition task may focus on incorporating one or more of these elements with the existing probabilistic lexical entailment model.

# Appendix A: Stop Words

Following is the list of 50 stop words that were filtered out of the probabilistic lexical entailment model.

| | | | |
|---|---|---|---|
| *the* | *he* | *or* | *do* |
| *of* | *as* | *an* | *if* |
| *and* | *by* | *were* | *more* |
| *to* | *you* | *her* | *when* |
| *a* | *at* | *she* | *who* |
| *in* | *are* | *we* | |
| *that* | *this* | *there* | |
| *is* | *not* | *been* | |
| *it* | *have* | *their* | |
| *was* | *had* | *one* | |
| *for* | *his* | *has* | |
| *on* | *from* | *will* | |
| *be* | *but* | *can* | |
| *with* | *they* | *all* | |
| *I* | *which* | *would* | |

# Appendix B: Pseudocode for the Textual Entailment Model

```
Function textualEntailmentModel(){

  READ input .xml file
  REPEAT{

    GET the next line

    IF the next line contains "<pair" {
      Set pair_id to the value of the id attribute in the line
      Set pair_value to the value of the value attribute in the line
      Set pair_task to the value of the task attribute in the line
    }
    ELSE IF the next line contains "<t>"{
      Erase all leading and trailing white space from the line
      Remove all instances of the characters ([{)]}"'`.,;:-!?
      Replace all digits with 0
      Convert all characters in the line to lowercase

      Create text array
      Set index to 0
      FOR each word in the remaining line {
        IF the stoplist does not contain the word {
          Set text(index) to that word
          Set index to index + 1
        }
      }
    }
    ELSE IF the next line contains "<h>" {
      Erase all leading and trailing white space from the line
      Remove all instances of the characters ([{)]}"'`.,;:-!?
      Replace all digits with 0
      Convert all characters in the line to lowercase

      Create hypothesis array
      FOR each word in the remaining line {
        IF the stoplist does not contain the word {
          Set hypothesis(index) to that word
          Set index to index + 1
        }
      }
      Create alignment array
      Create scores array
      Set hypothesis_index to 0
      FOR each word in the hypothesis array {
        Set h_word to the current hypothesis word
        Set max to 0
```

```
        Set overall_score to 1
        Set align_word to NULL

        IF any word from the text array equals h_word {
          Set max to 1
          Set align_word to h_word
        }
        ELSE {
          FOR each word in the text array {
            Set t_word to the current text word
            IF LEP(h_word, t_word) > max {
              Set max to LEP(h_word, t_word)
              Set align_word to t_word
            }
          }
        }
        Set overall_score to overall_score * max
        Set scores(hypothesis_index) to max
        Set alignment(hypothesis_index) to align_word
        Set hypothesis_index to hypothesis_index + 1
      }
    }
    ELSE IF the next line contains "</pair>" {
      APPEND results to output file
    }
  }UNTIL the end of the input file is reached
}
```

# Appendix C: Pseudocode for calculating $LEP_{co}(u,v)$

```
Function LEPco(u,v){

  Set v_query to v

  IF u precedes v alphabetically
    Set uv_query to "u"+_+"v"
  ELSE
    Set uv_query to "v"+_+"u"

  IF cache contains v_query
    GET number of hits for v_query from cache
    Set denominator to hits
  ELSE
    Set v_hits to webSearch(v_query)
    Add (v_query, v_hits) to cache
    Set denominator to v_hits

  IF cache contains uv_query
    GET number of hits for uv_query from cache
    Set numerator to hits
  ELSE
    Set uv_hits to webSearch(uv_query)
    Add (uv_query, uv_hits) to cache
    Set numerator to uv_hits

  Set LEPuv = numerator / denominator

  RETURN LEPuv
}
```

# Appendix D: Java code for counting word-feature pairs

```
/**********************************************************************
 * PairCounts takes in the names of 2 files: a master count file, listing
 * words followed by a list of features they co-occur with and relevant
 * count; and a filenames list of parse files yet to be processed.
 *
 * It updates the master count file after each set of 10 parsed files.
 **********************************************************************/

import java.io.*;
import java.util.regex.*;
import java.util.*;


public class PairCounts {


    public PairCounts() {
    }

    // the following ArrayList will be used to index all words
    private static ArrayList vocab;

    /** These Hashtables of ArrayLists will store the features applicable
     *  to each word and the count of each word/feature pair. They will
     *  share the same index so that the order of features in the
     *  ArrayList for word1 stored in words2features will correspond to
     *  the order of the counts in the ArrayList indexed by word1 stored
     *  in words2wfcounts.
     */
    private static Hashtable words2features;
    private static Hashtable words2wfcounts;

    private static int total_feature_count;

    public static Pattern p = Pattern.compile("_\\d+");
    private static Pattern nonalphanum = Pattern.compile("[\\W&&[^-]]");
    private static Pattern num = Pattern.compile("\\d");

    public static void main(String[] args) {

        /************************************************
         * Read args
         ************************************************/
        String wfpairsfile = args[0];
```

```
String listfile = args[1];
int groupsize=10;
if (args.length > 2)
    groupsize = Integer.valueOf(args[2]);

/**************************************************
 * Keep running program until the list of input
 * files is exhausted
 **************************************************/
do {
    /**************************************************
     * Zero indexes
     **************************************************/
    System.out.println("Initialising index...");
    zero_indexes();

    /**************************************************
     * Read list of input files
     **************************************************/

    System.out.print("Reading input files...");
    ArrayList filenames = new ArrayList();

    try {
        BufferedReader fin =
            new BufferedReader(new InputStreamReader
                (new FileInputStream
                (listfile)));
        if (!fin.ready()) {
            System.out.println("End of file list reached");
            return;
        }

        while (fin.ready()) {
            String name = fin.readLine();
            filenames.add(name);
        }

        fin.close();
    }catch(IOException e) {
        System.out.println(e);
    }
    System.out.print("Done\n");

    /**************************************************
     * Get names of next (10) files to process
     **************************************************/
    int arraysize = 0;
    String[] filelist = new String[groupsize];
    for (int jj=0; jj<groupsize; jj++)
        if (!filenames.isEmpty()) {
        filelist[jj] = filenames.get(0).toString();
        filenames.remove(0);
        arraysize++;
        }
```

```
/************************************************
 * Process each of (10) files and update counts
 ************************************************/
for (int jjj=0; jjj<arraysize; jjj++) {
    try {
        String currentfilename = filelist[jjj];
        // read in the parse file
        BufferedReader in =
            new BufferedReader(new InputStreamReader
                (new FileInputStream
                (currentfilename)));
        System.out.println("Processing: " + currentfilename);
        System.out.println("Vocab size: " + vocab.size());

        if (!in.ready()) {
            System.out.println
                    ("File " + currentfilename
                     + " did not open");
            continue;
        }

        /************************************************
         * Read in word/feature pairs
         ************************************************/
        // go through parsed results line-by-line
        while (in.ready()) {
            String line = in.readLine().toLowerCase();

            // ignore lines starting with <c> and containing
            // s_0
            if (line.contains("<c>") || line.contains("s_0"))
                continue;
            // ignore blank lines
            if (!line.contains("("))
                continue;
            // ignore commented lines
            if (line.contains("#"))
                continue;

            // filter out the position markers like _13
            Matcher m = p.matcher(line);
            line = m.replaceAll("");

            // change all digits to 0
            Matcher m4 = num.matcher(line);
            line = m4.replaceAll("0");

            // filter out the opening and closing brackets
            line = line.substring(1,line.length()-1);

            String[] lineargs = line.split("\\s");
            for (int k=0; k<lineargs.length; k++)
                lineargs[k] = lineargs[k].trim();
```

```
String GR, w1, w2, w1feature, w2feature;

// ignore lines with less than 3 fields
if (lineargs.length <3)
    continue;

// if there are more than 3 fields, check GR type
// and append optional fields to GR type, and then
// reduce all fields to the first three positions
// in the array
if (lineargs.length > 3) {
    if (lineargs[0].equals("ncsubj") ||
         lineargs[0].equals("xsubj") ||
         lineargs[0].equals("csubj"))
        lineargs[0] = lineargs[0] + "." +
            lineargs[3];

    else {
        lineargs[0] = lineargs[0]+ "." +
            lineargs[1];
        lineargs[1] = lineargs[2];
        lineargs[2] = lineargs[3];
    }
}

// pull out GR, w1 and w2
GR = lineargs[0];
w1 = lineargs[1];
w2 = lineargs[2];

// Ignore words with non-alphanumeric characters
Matcher m2 = nonalphanum.matcher(w1);
if (m2.find()) {
    continue;
}
Matcher m3 = nonalphanum.matcher(w2);
if (m3.find()) {
    continue;
}

// create two features - one for each word
// D denotes that the word inside brackets was the
// dependent in the original parse, whereas H
// denotes that the word inside brackets was the
// head in the original parse
w1feature = "< " + GR + " " + w2 + " D >";
w2feature = "< " + GR + " " + w1 + " H >";


/*************************************************
 * Add the two new features to the indexes

*************************************************/
add_to_index(w1, w1feature, 1);
add_to_index(w2, w2feature, 1);
```

```
            }
            in.close();
            System.out.print("Done\n");
        } catch (IOException e) {
            System.out.println(e);
        }

        /*****************************************
         * Write Word/Feature Counts to temporary file
         *****************************************/

        // make sure they come out in alphabetical order
        Collections.sort(vocab);
        System.out.print
             ("Writing word/feature counts to temporary file...");
        try {
            FileWriter wfpairs_out =
                    new FileWriter("tempfile.txt");

            // Write total feature count
            wfpairs_out.write(total_feature_count+"\n");

            // for each word in vocab, write features and counts
            for (int t=0; t<vocab.size(); t++) {
                String thisword = vocab.get(t).toString();

                // write word
                wfpairs_out.write(thisword + " ");

                // write features and counts
                ArrayList currentfeaturelist =
                        (ArrayList) words2features.get(thisword);
                ArrayList currentcountlist =
                        (ArrayList) words2wfcounts.get(thisword);

                for (int q=0; q<currentfeaturelist.size(); q++) {

                    String currentfeature =
                            currentfeaturelist.get(q).toString();
                    int paircount =
                            Integer.valueOf
                            (currentcountlist.get(q).toString());

                    wfpairs_out.write(currentfeature + " "
                            + paircount + " ");
                }
                wfpairs_out.write("\n");
            }
            wfpairs_out.close();
        } catch (IOException j) {
            System.out.println(j);
        }
        System.out.print("Done\n");
    }
```

```
        /*************************************************
         * Merge temporary counts file with large file
         *************************************************/
        System.out.print("Merging files...");
        String[] mergeargs =
            {"tempfile.txt", wfpairsfile, "tempmergedfile.txt"};
        MergeCounts.main(mergeargs);

        // copy temporary merged file back to wfpairsfile
        try{
            BufferedReader tempin =
              new BufferedReader(new InputStreamReader
                    (new FileInputStream
                    ("tempmergedfile.txt")));

            FileWriter tempout = new FileWriter(wfpairsfile);
            while (tempin.ready()) {
                String linein = tempin.readLine();
                tempout.write(linein+"\n");
            }
            tempin.close();
            tempout.close();
        }catch(IOException e){
            System.out.println(e);
        }
        System.out.print("Done\n");

        /*************************************************
         * Output remaining files to file list
         *************************************************/
        System.out.print("Writing remaining files to file list..");
        try {
            FileWriter filenames_out =
                    new FileWriter(listfile);
            if (filenames.size() > 1){
                for (int i=0; i<filenames.size(); i++)
                    filenames_out.write
                       (filenames.get(i).toString() + "\n");
            }
            filenames_out.close();
            filenames = null;
        }  catch (IOException g) {
            System.out.println(g);
        }
        System.out.print("Done\n");
    }while (1==1);
}
```

```
    /*************************************************************
     * This function takes in a word and feature pair with count x and
     * adjusts the count of this word/feature pair in the Hashtables and
     * ArrayList accordingly.
     *************************************************************/
    private static void add_to_index(String word, String feature, int x) {

        // add x to the total feature count
        total_feature_count += x;
        // if the word is new, add it to the vocab list
        int i = vocab.indexOf(word);
        if (i == -1) {
            vocab.add(word);
            words2features.put(word, new ArrayList());
            words2wfcounts.put(word, new ArrayList());
        }
        // check whether the word/feature pair has been encountered yet
        ArrayList tempfeaturelist = (ArrayList) words2features.get(word);
        ArrayList tempcountlist = (ArrayList) words2wfcounts.get(word);
        int m = tempfeaturelist.indexOf(feature);
        // if it's new, add the feature to the word's ArrayList with
        // count of x
        if (m == -1) {
            tempfeaturelist.add(feature);
            tempcountlist.add(x);

            words2features.remove(word);
            words2features.put(word, tempfeaturelist);
            words2wfcounts.remove(word);
            words2wfcounts.put(word, tempcountlist);
        }
        // otherwise increase the feature's count by x
        else {
            int n = Integer.valueOf(tempcountlist.get(m).toString());
            n+=x;
            tempcountlist.remove(m);
            tempcountlist.add(m,n);

            words2features.remove(word);
            words2features.put(word, tempfeaturelist);
            words2wfcounts.remove(word);
            words2wfcounts.put(word, tempcountlist);
        }
        tempfeaturelist = null;
        tempcountlist = null;
    }
    private static void zero_indexes() {
        vocab = new ArrayList();
        words2features = new Hashtable();
        words2wfcounts = new Hashtable();
        total_feature_count = 0;
    }
}
```

# Appendix E: Java code for calculating feature weights

```
/**********************************************************************
 * MIWeights takes in the names of 2 files: one is the master list of
 * words followed by their features and counts; the other is a feature
 * index of features followed by their total count and wordset. It
 * processes these files to create an index of MI weights for word/feature
 * pairs with the format:
 *
 * word1 feature1 MIweight1 feature2 MIweight2 ...
 * word2 feature1 MIweight1 feature2 MIweight2 ...
 *
 **********************************************************************/

import java.io.*;
import java.util.regex.*;
import java.util.*;
import java.lang.Math;

public class MIWeights {

    public MIWeights() {
    }

    // The following ArrayLists contain a list of features and their
    // respective total counts
    private static ArrayList features;
    private static ArrayList featurecounts;

    private static int total_feature_count;

    public static void main(String[] args) {

        // read in args
        String wfpaircountsfile = args[0];
        String featureindex = args[1];
        String MIweightsfile = args[2];

        // populate feature lists
        try {
            BufferedReader fin = new BufferedReader(new InputStreamReader
                    (new FileInputStream
                    (featureindex)));
            if (!fin.ready()) {
                System.out.println("featureindex file not ready");
                return;
```

```
        }
        System.out.println("Populating feature lists");

        features = new ArrayList();
        featurecounts = new ArrayList();

        // read in features and counts one at a time
        while (fin.ready()) {
            String line = fin.readLine();
            String[] lineparts = line.split(" ");

            String feature = lineparts[0] + " " + lineparts[1]
                    + " " + lineparts[2] + " " + lineparts[3]
                    + " " + lineparts[4];
            int count = Integer.valueOf(lineparts[5]);
            features.add(feature);
            featurecounts.add(count);
            System.out.println("Feature: "
                               + feature + "  Count: " + count);
        }
        fin.close();
    }catch (IOException e) {
        System.out.println(e);
    }
    // process wfpair counts file one line at a time and output
    // calculated MI weights to MI weights file
    try {
        BufferedReader wfin = new BufferedReader(new InputStreamReader
                (new FileInputStream
                (wfpaircountsfile)));
        if (!wfin.ready()) {
            System.out.println("wfpairs file not ready");
            return;
        }
        System.out.println("Reading from file: " + wfpaircountsfile);
        FileWriter weights_out =
                new FileWriter(MIweightsfile);

        // The first line of this file is the total feature count
        total_feature_count = Integer.valueOf(wfin.readLine());
        System.out.println("Total Feature Count: "
                           + total_feature_count);

        int counter = 0;
        // read in words, features and counts one at a time
        while (wfin.ready()) {
            counter++;
            String line = wfin.readLine();
            String[] lineparts = line.split(" ");

            String word = lineparts[0];
            System.out.print(counter+" "+word+"...");
            weights_out.write(word + " ");

            ArrayList currentfeaturelist = new ArrayList();
            ArrayList currentfeaturecounts = new ArrayList();
```

```
// read in word/feature pair counts
for (int j=1; j<lineparts.length; j+=6) {
    String feature = lineparts[j] + " " +
            lineparts[j+1] + " " +
            lineparts[j+2] + " " +
            lineparts[j+3] + " " +
            lineparts[j+4];
    int count = Integer.valueOf
                    (lineparts[j+5].toString());
    currentfeaturelist.add(feature);
    currentfeaturecounts.add(count);
}
// calculate total count for the current word
int cw = 0;
for (int k=0; k<currentfeaturecounts.size(); k++)
    cw += Integer.valueOf
            (currentfeaturecounts.get(k).toString());

// now, for each feature in the currentfeaturelist,
// calculate the MI weight and store it in an ArrayList
ArrayList currentfeatureweights = new ArrayList();
for (int m=0; m<currentfeaturelist.size(); m++) {
    String f = currentfeaturelist.get(m).toString();
    int cf = Integer.valueOf(featurecounts.get
            (features.indexOf(f)).toString());
    int cwf = Integer.valueOf
            (currentfeaturecounts.get(m).toString());

    double weight = 0;

    // numerator = count(w,f) * total_feature_count
    double numerator = cwf * total_feature_count;

    // denominator = count(w) * count(f)
    double denominator =  cw * cf;

    double intermediate = 1/denominator;
    intermediate = intermediate*cwf*total_feature_count;

    weight = Math.log(intermediate)/Math.log(2);
    currentfeatureweights.add(m, weight);
}

// print feature sum, and features w/ weights to file
weights_out.write(cw + " ");
for (int nn=0; nn<currentfeaturelist.size(); nn++) {
    double wt = Double.valueOf(currentfeatureweights.
                    get(nn).toString());
    if (!Double.isNaN(wt))
        weights_out.write
          (currentfeaturelist.get(nn).toString()
           + " " + wt + " ");
}
weights_out.write("\n");
```

```
                currentfeaturelist = null;
                currentfeaturecounts = null;
                currentfeatureweights = null;
                System.out.print("Done\n");
            }
            wfin.close();
            weights_out.close();
        }catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

# Appendix F: Java code for calculating $LEP_{ws}$ rankings

```
import java.io.*;
import java.util.*;

public class RankSims {

    // the active features list will be utilized for filtering out any
    // features with total count less than the feature_filter from the sim
    // calculation
    private static ArrayList activefeatures;
    private static int feature_filter = 10;

    // this filter will act to ignore any word/feature pairs with weight
    // less than the given amount
    private static int weight_filter = 4;

    private static ArrayList vocab;

    // This ArrayList holds the sums of all feature weights for each word
    // in the vocabulary. It shares its index with the vocab ArrayList.
    private static ArrayList featuresums;

    // the stopwords list contains the 50 most common words from the
    // corpus, plus '-'
    private static String[] stopwords = {"the","of","and","to","a","in",
      "that","is","it","was","for","on","be","with","I","he","as","by",
      "you","at","are","this","not","have","had","his","from","but",
      "they","which","or","an","were","her","she","we","there","been",
      "their","one","has","will","can","all","would","do","if","more",
      "when","who","-"};

    // the stopfeatures list contains the 40 most common features from the
    // corpus in terms of inverse word frequency; all features in this
    // list occur with over 10,000 words
    private static String[] stopfeatures = {"< A:lex-mod:U - D >",
      "< N:nn:N the D >","< N:lex-mod:U - D >","< N:nn:N of D >",
      "< N:nn:N and D >","< N:nn:N a D >","< N:nn:N to D >",
      "< N:nn:N the H >","< N:nn:N in D >","< N:nn:N and H >",
      "< N:nn:N to H >","< N:nn:N with D >","< N:nn:N by D >",
      "< N:nn:N in H >","< N:nn:N of H >","< N:nn:N for D >",
      "< N:nn:N a H >","< N:nn:N as D >","< N:nn:N from D >",
      "< N:nn:N that D >","< N:nn:N is D >","< N:nn:N on D >",
      "< N:nn:N was D >","< N:nn:N at D >","< N:nn:N is H >",
      "< N:mod:A the H >","< N:nn:N or D >","< N:nn:N was H >",
      "< N:nn:N it D >","< N:nn:N be D >","< N:mod:A and H >",
      "< N:nn:N that H >","< N:nn:N his D >","< N:nn:N he D >",
      "< N:nn:N on H >","< N:nn:N are H >","< N:nn:N for H >",
```

```
  "< N:nn:N an D >","< N:nn:N i D >","< N:mod:A of H >"};

// the featurewords and featureweights Hashtables hold ArrayLists of
// words and their weights respectively, indexed by feature
private static Hashtable featurewords;
private static Hashtable featureweights;

public RankSims() {
}

/*****************************************************************
 * This function takes four arguments - the name of a word/feature
 * pairs weights file, the name of a feature index, the name of a
 * feature weights index, and the
 * name of the file to output the similarity scores to.
 *
 * It outputs a file with one word and its similarity scores
 * per line, in descending order of similarity.
 *****************************************************************/
public static void main(String[] args) {

    // read in arguments
    feature_filter = Integer.valueOf(args[0]);

    weight_filter = Integer.valueOf(args[1]);

    String wfweightsfile = args[2];
    String featureindex = args[3];
    String featureweightsindex = args[4];
    String simsfile = args[5];

    // include a file to write featuresums out to
    String featuresumfile = args[6];

    // include the option to start from where we left off
    boolean start = true;
    String lastword = null;
    if (args.length >= 8) {
        lastword = args[7];
        start = false;
    }

    // include the option to read words we want the sim scores from
    // from a file
    String fileofwords = null;
    boolean readwordsfromfile = false;
    ArrayList wordlist = new ArrayList();
    if (args.length > 8) {
        readwordsfromfile = true;
        fileofwords = args[8];
        try {
            BufferedReader wordsin =
                new BufferedReader(new InputStreamReader
                    (new FileInputStream
                    (fileofwords)));
            System.out.println("Generating word list");
```

```
        while (wordsin.ready()){
            wordlist.add(wordsin.readLine().trim());
        }
        wordsin.close();
    }catch (IOException e) {
        System.out.println(e);
    }
}

// initialize vocabulary and feature sums lists
vocab = new ArrayList();
featuresums = new ArrayList();
activefeatures = new ArrayList();
featurewords = new Hashtable();
featureweights = new Hashtable();

// populate the active features list with all features whose count
// is greater than the feature_filter
getactivefeatures(featureindex);

// populate vocab and featuresums lists from wfweights file
getvocab(wfweightsfile, featuresumfile);

// populate featurewords and featureweights hashtables from the
// feature weight index file
getfeaturelists(featureweightsindex);

// calculate similarity scores for each word in the vocab, from
// start of the file to finish
try {
    BufferedReader fin2 = new BufferedReader(new InputStreamReader
            (new FileInputStream
            (wfweightsfile)));

    System.out.println("Calculating similarity scores");

    // for each word w...
    while (fin2.ready()) {
        String line = fin2.readLine();
        String[] lineparts = line.split(" ");

        String w = lineparts[0];
        if (!vocab.contains(w))
            continue;

        if (start == false) {
            if (w.equals(lastword)) {
                start = true;
                continue;
            } else
                continue;
        }

        if (readwordsfromfile == true) {
            if (!wordlist.contains(w))
```

```
        continue;
}

// remove the current word from the wordlist so we'll know
// which words are left over at the end
wordlist.remove(w);

System.out.print(w + "...");

int w_index = vocab.indexOf(w);

// store w's feature sum locally
double w_featuresum =
      Double.valueOf(featuresums.get(w_index).toString());

// create array to hold the similarity scores for other
// words v and set all values to 0 initially
ArrayList simwords = new ArrayList();
ArrayList simscores = new ArrayList();

// create array to hold the featureweights for all similar
// words locally
ArrayList simsums = new ArrayList();

// create arrays to hold w's active features and their
// weights
ArrayList wfeatures = new ArrayList();
ArrayList wfeatureweights = new ArrayList();

// for each feature f in F(w)...
for (int i=2; i<lineparts.length; i+=6) {
    String f = lineparts[i] + " " +
            lineparts[i+1] + " " +
            lineparts[i+2] + " " +
            lineparts[i+3] + " " +
            lineparts[i+4];

    // make sure the feature is in the active feature list
    if (!activefeatures.contains(f))
        continue;

    double wf_weight =
            Double.valueOf(lineparts[i+5]);

    // make sure this feature's weight is higher than the
    // filter
    if (wf_weight<weight_filter||Double.isNaN(wf_weight))
        continue;

    wfeatures.add(f);
    wfeatureweights.add(wf_weight);
}

// Now, for each feature in wfeatures, find WS(f) and
// update the similarity score between w and all
// v(elem of)WS(f) get WS(f) for each feature in F(w)
```

```
for (int j=0; j<wfeatures.size(); j++) {
    String feature = wfeatures.get(j).toString();
    double wfweight =
        Double.valueOf(wfeatureweights.get(j).toString());
    if (!featurewords.containsKey(feature))
        continue;
    ArrayList words =(ArrayList)featurewords.get(feature);
    ArrayList weights =
        (ArrayList)featureweights.get(feature);
    System.out.println(w + " : " + feature);
    for (int k=0; k<words.size(); k++) {
        String v = words.get(k).toString();
        if (v.equals(w))
            continue;

        double vwt =
              Double.valueOf(weights.get(k).toString());
        //int v_simscoresindex = vocab.indexOf(v);
        // find the word v in w's simscores index
        int v_simscoresindex = simwords.indexOf(v);
        if (v_simscoresindex == -1) {
            simwords.add(v);
            simscores.add(wfweight + vwt);
            simsums.add
              (featuresums.get(vocab.indexOf(v)));

        } else {
            double score = Double.valueOf
              (simscores.get
              (v_simscoresindex).toString());
            score = score + wfweight + vwt;
            simscores.remove(v_simscoresindex);
            simscores.add(v_simscoresindex, score);
        }
    }
    words = null;
    weights = null;
}

// divide each value in the simscores array by
// featuresum(w) + featuresum(v)
System.out.println("Dividing sim values");
for (int s=0; s<simwords.size(); s++) {
    Object v = simwords.get(s);

    double vfeaturesum = Double.valueOf
                         (simsums.get(s).toString());

    double score = Double.valueOf
                         (simscores.get(s).toString());
    score = score / (w_featuresum + vfeaturesum);
    simscores.remove(s);
    simscores.add(s, score);
}
```

```
            // order simscores high to low, with the lowest being 0.04
            System.out.println("Ordering sim values");
            ArrayList inordersimwords = new ArrayList();
            ArrayList inordersimscores = new ArrayList();

            for (int t=0; t<simwords.size(); t++) {
                String word = simwords.get(t).toString();
                double sim = Double.valueOf
                                (simscores.get(t).toString());
                if (sim < 0.04)
                    continue;
                if (inordersimwords.isEmpty()) {
                    inordersimwords.add(word);
                    inordersimscores.add(sim);
                } else {
                    boolean added = false;
                    for (int x=0; x<inordersimwords.size(); x++) {
                        double current = Double.valueOf
                                (inordersimscores.get(x).toString());
                        if (sim > current) {
                            inordersimwords.add(x, word);
                            inordersimscores.add(x, sim);
                            added = true;
                            break;
                        }
                    }
                    if (added==false) {
                        inordersimwords.add(word);
                        inordersimscores.add(sim);
                    }
                }
            }

            // print simscores over .04 to file
            System.out.println("Writing sim values to file");
            FileWriter printsims = new FileWriter(simsfile, true);
            printsims.write(w + " ");
            System.out.println(inordersimwords.size());
            for (int y=0; y<inordersimwords.size(); y++) {
                printsims.write(inordersimwords.get(y)
                    + " " + inordersimscores.get(y) + " ");
            }
            printsims.write("\n");
            printsims.close();
        }
        fin2.close();
    }catch (IOException e) {
        System.out.println(e);
    }

}

private static void getactivefeatures(String featureindex) {
    try {
        /*************************************************************
         * Begin by creating the active features list by reading
```

```
         * through the feature index and including any features with
         * total count less than the filter.
         ************************************************************/

        // read in the feature index file
        System.out.println("Reading features");
        BufferedReader featuresin =
              new BufferedReader(new InputStreamReader
                (new FileInputStream
                (featureindex)));
        if (!featuresin.ready())
            throw new IOException();
        // put stop features in arraylist
        ArrayList stopfeaturelist = new ArrayList();
        for (int ii=0; ii<stopfeatures.length; ii++) {
            stopfeaturelist.add(stopfeatures[ii]);
        }
        // go through features line-by-line
        while (featuresin.ready()) {
            String line = featuresin.readLine();
            String[] lineparts = line.split(" ");

            String feature = lineparts[0] + " " + lineparts[1]
                    + " " + lineparts[2] + " " + lineparts[3]
                    + " " + lineparts[4];
            if (stopfeaturelist.contains(feature))
                continue;

            int featurecount = Integer.valueOf(lineparts[5]);


            if (featurecount >= feature_filter) {
                System.out.println(feature);
                activefeatures.add(feature);
            }
        }
        featuresin.close();
        System.out.println("Active feature list complete");
    } catch (IOException e) {
        System.out.println(e);
    }
}

// This function reads in the vocab from the miweights file and
// calculates each word's feature sum from the active features. The
// miweights file is already filtered to include only words for which
// count(w) > 6.
private static void getvocab
  (String wfweightsfile, String featuresumfile) {

    // get all the vocab from the featuresums file
    try {
        System.out.println
              ("Populating vocab lists from feature sums file");
        BufferedReader sumsin =
```

```
              new BufferedReader(new InputStreamReader
                (new FileInputStream
                (featuresumfile)));
            while (sumsin.ready()) {
                String line = sumsin.readLine();
                String[] lineparts = line.split(" ");
                String word = lineparts[0];

                boolean stop = false;
                for (int j=0; j<stopwords.length; j++) {
                    if (word.equals(stopwords[j])) {
                        stop = true;
                        break;
                    }
                }
                if (stop) continue;

                System.out.println(lineparts[0]);
                vocab.add(lineparts[0]);
                featuresums.add(lineparts[1]);
            }
            sumsin.close();
        }catch (IOException e) {
            System.out.println(e);
        }
    }

    // this function populates the featurewords and featureweights
    // hashtables
    private static void getfeaturelists(String featureweightsindex) {
        try {
            BufferedReader fwindex =
                  new BufferedReader(new InputStreamReader
                    (new FileInputStream
                    (featureweightsindex)));
            System.out.println
                  ("Populating feature word and weight lists");
            while (fwindex.ready()) {
                String fline = fwindex.readLine();
                String[] flineparts = fline.split(" ");

                String feature = flineparts[0] + " "
                        + flineparts[1] + " "
                        + flineparts[2] + " "
                        + flineparts[3] + " "
                        + flineparts[4];
                if (!activefeatures.contains(feature))
                    continue;
                System.out.println(feature);
                ArrayList words = new ArrayList();
                ArrayList weights = new ArrayList();
                for (int j=5; j<flineparts.length; j+=2) {
                    String v = flineparts[j];

                    // keep from including terms that are not in the vocab
                    if (!vocab.contains(v))
```

```
                        continue;

                    double vwt = Double.valueOf(flineparts[j+1]);

                    // check to make sure the weight of this feature
                    // and v is larger than the weight filter
                    if (vwt >= weight_filter) {
                        words.add(v);
                        weights.add(vwt);
                        System.out.println(feature + " : " + v);
                    }
                }
                featurewords.put(feature, words);
                featureweights.put(feature, weights);
            }
            fwindex.close();
        }catch(IOException e) {
            System.out.println(e);
        }

    }
}
```

# Bibliography


Bar-Haim, R., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B., et al. (2006). The Second PASCAL Recognising Textual Entailment Challenge. *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*. Venice.

BNC Consortium. (2001). The British National Corpus, version 2 (BNC World).

Brown, P. F., Cocke, J., Della Pietra, S., Della Pietra, V. J., Jelinek, F., Lafferty, J. D., et al. (1990). A statistical approach to machine translation. *Computational Linguistics, 16* (2), 79-85.

Chen, S. F., & Goodman, J. (1996). An Empirical Study of Smoothing Techniques for Language Modeling. *Proceedings of the 34th annual meeting on Association for Computational Linguistics* (pp. 310-318). Santa Cruz: Association for Computational Linguistics.

Dagan, I. (2000). Contextual Word Similarity. In R. Dale, H. Moisl, & H. Somers (Eds.), *Handbook of Natural Language Processing* (pp. 459-476). Marcel Dekker Inc.

Dagan, I., Glickman, O., & Magnini, B. (2006). The PASCAL Recognising Textual Entailment Challenge. In Q.-C. e. al (Ed.), *MLCW 2005, LNAI Volume 3944* (pp. 177-190). Springer-Verlag.

de Salvo Braz, R., Girju, R., Punyakanok, V., Roth, D., & Sammons, M. (2005). An inference model for semantic entailment in natural language. *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*.

Dolan, W. B., Quirk, C., & Brockett, C. (2004). Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. *Proceedings of the 20th international conference on Computational Linguistics* (p. Article no. 350). Geneva, Switzerland: Association for Computational Linguistics.

Geffet, M., & Dagan, I. (2004). Feature vector quality and distributional similarity. *Proceedings of COLING 2004*, (pp. 247-253). Geneva.

Geffet, M., & Dagan, I. (2005). The distributional inclusion hypotheses and lexical entailment. *Proceeding sof the 43rd Annual Meeting of the Association for Computational Linguistics (ACL '05)* (pp. 107-114). Ann Arbor, Michigan: Association for Computational Linguistics.

Glickman, O. (2006, June). Applied textual entailment. *Ph.D. thesis* . Ramat Gan, Israel: Bar Ilan University.

Glickman, O., & Dagan, I. (2005). A Probabilistic Setting and Lexical Cooccurrence Model for Textual Entailment. *Proceedings of the ACL Workshop on Empirical Modeling of Semantic Equivalence and Entailment* (pp. 43-48). Ann Arbor, Michigan: Association for Computational Linguistics.

Glickman, O., Dagan, I., & Koppel, M. (2005). A Probabilistic Classificaiton Approach for Lexical Textual Entailment. *Twentieth National Conference on Artificial Intelligence (AAAI-05).* Pittsburgh, Pennsylvania: Association for the Advancement of Artificial Intelligence.

Glickman, O., Dagan, I., & Koppel, M. (2005). Web based probabilistic textual entailment. *Proceedings of the PASCAL Recognising Textual Entailment Challenge Workshop*.

Harris, Z. S. (1954). Distributional structure. *Word , 10* (23), 146-162.

Koehn, P., Och, F. J., & Marcu, D. (2003). Statistical phrase-based translation. *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1* (pp. 48-54). Edmonton, Canada: North American Chapter of the Association for Computational Linguistics.

Koehn, P., Och, F. J., & Marcu, D. (2003). Statistical phrase-based translation. *Proceedings of the 2003 Conference of the North American Chapter of the Association*

*for Computational Linguistics on Human Language Technology - Volume 1* (pp. 48-54). Edmonton, Canada: Association for Computational Linguistics.

Landis, J. R., & Koch, G. G. (1997). The measurements of observer agreement for categorical data. *Biometrics , 33*, 159-174.

Lin, D. (1998). Automatic retrieval and clustering of similar words. *Proceedings of COLING-ACL98*. Montreal.

Lin, D. (n.d.). *Downloads*. Retrieved August 22, 2007, from Dekang Lin's Home Page: http://www.cs.ualberta.ca/~lindek/downloads.htm

Nielsen, R., Ward, W., & Martin, J. H. (2006). Toward dependency path based entailment. *Proceedings of the Second PASCAL Recognising Textual Entailment Challenge Workshop*. Venice.

Perez, D., & Alfonseca, E. (2005). Application of the Bleu algorithm for recognising textual entailments. Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment.

Quirk, C., Brockett, C., & Dolan, W. B. (2004). Monolingual machine translation for paraphrase generation. *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing* (pp. 142-149). Barcelona, Spain: Association for Computational Linguistics.

Rennie, J. D. (2004, Febuary 19). *Derivation of the F-measure*. Retrieved August 2, 2007, from MIT Computer Science and Artificial Intelligence Laboratory: http://people.csail.mit.edu/jrennie/writing/fmeasure.pdf

Roth, M. (2003, December 29). *Java Sizeof()*. Retrieved August 23, 2007, from Martin's Java Notes: http://martin.nobilitas.com/java/sizeof.html

Tatu, M., & Moldovan, D. (2005). A semantic approach to recognizing textual entailment. *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*.

van Rijsbergen, C. J. (1979). *Information Retrieval*. London: Butterworths.

Witten, I. H., & Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques* (2 ed.). San Francisco: Morgan Kaufman.