Feature Analysis for Paraphrase Recognition and Textual Entailment

Andrea Segura-Olivares, Alejandro García, and Hiram Calvo

Centro de Investigación en Computación (CIC),
Instituto Politécnico Nacional (IPN), Mexico City, Mexico
msegura_b12@sagitario.cic.ipn.mx,
igarcia_b12@sagitario.cic.ipn.mx, hcalvo@cic.ipn.mx

Abstract. Paraphrase recognition is the task of Natural Language Processing of detecting if an expression restated as another expression contains the same information. Textual Entailment recognition, while being similar to paraphrase recognition, is a task that consists in finding out if a given text can be observed as a consequence of another text fragment, sometimes considering only part of the original meaning, or adding some inferences based on common sense. Traditionally, for solving this problem, several *lexical, syntactic* and *semantic* based techniques are used. In this work, we seek to use the less resources as possible, while being effective. For this, we perform a feature analysis for performing Paraphrase Recognition and recognizing Textual Entailment experimenting with the combination of several Natural Language Processing techniques like *word overlapping, syntactic analysis*, and *elimination of stop words*. Particularly, we explore using the syntactic n-grams technique combined with some auxiliary approaches such as *stemming, synonym detection, similarity measures* and *linear interpolation*. We measure and compare the performance of our system by using the Microsoft Research Paraphrase Corpus, and the RTE-3 test set for Paraphrasing and Textual Entailment, respectively. Syntactic n-grams produce good results for Paraphrase Recognition. As far as we know, syntactic n-grams had not been used for this task. For Textual Entailment, our best results were obtained by using a simple word overlapping algorithm based on stemming and elimination of stop words.

1 Introduction

The study of new techniques in Natural Language Processing (NLP) has become more popular recently between researchers. As a consequence, there are several tasks that are considered solved on this important area; however, NLP still has some challenges that need to be solved. Some of them are Paraphrase Recognition and Textual Entailment.

The best results reported for these tasks usually have one characteristic in common, and it is that they require costly resources such as combinations of lexical analysis, semantics, logic inference, background knowledge and machine learning [10].

For languages like English, having the required resources may not be a problem because English has been studied for several people, and there are many NLP tools

⋆ Work done under support of CONACyT-SNI, SIP-IPN, COFAA-IPN, and PIFI-IPN

that can be used for the purpose of this kind of systems. However, for other languages that have not been studied deeply, resources could be a restriction for implementing NLP systems. Therefore, the purpose of our work was to create a system that employed as less resources as possible but still having a good performance on the Paraphrase Recognition and Textual Entailment tasks.

In the following sections we will describe two standard tasks for Paraphrase Recognition, and Textual Entailment, as well as the general methods for tackling them. For this work we experimented with several NLP Techniques (See Section 2), and particularly we propose using syntactic n-grams, described in Section 2.3. We present our experiments and their evaluation with two respective gold standards in Section 3 for Paraphrase Recognition, and in Section 4 for Textual Entailment. Finally we draw our conclusions in Section 5.

Paraphrase Recognition Paraphrasing is the process of restating an expression E_1 in other expression(s) E_2, E_3, ..., E_n that convey the same meaning. For instance, the expression:

- $\mathbf{S}_1$: Juan Rulfo wrote "El llano en llamas".
- $\mathbf{S}_2$: "El llano en llamas" was written by Juan Rulfo.

are paraphrases of each other.

In general the paraphrase processing is divided in three main tasks that are described below:

1. **Extraction**: this task has the goal of obtaining a set as large as possible of pairs (S_1, S_2) that conform a paraphrase pair from a big corpus given as input of the system.
2. **Generation**: this task has the objective to yield a set of expressions (S_2, S_3, ..., S_n) as large as possible that are paraphrases of the input string S_1.
3. **Recognition**: this task has as objective to detect if two given expressions (S_1, S_2) given are paraphrases of each other.

Textual Entailment Textual Entailment recognition is the task of finding out whether the semantics of a text can be inferred from the semantics of another text. The entailing and entailed text are termed *text (T)* and *hypothesis (H)* respectively. An example of textual entailment is the following:

- T: The drugs that slow down Alzheimer's disease work best the earlier you administer them.
- H: Alzheimer's disease can be slowed down using drugs.

where we can see that H can be inferred from T. Therefore, this is a true textual entailment pair.
An example of a non textual entailment pair is the following:

- T: It is important to stress that this is not a confirmed case of rabies.
- H: A case of rabies was confirmed.

where the semantics of the text H cannot be inferred from the text T.

Differently to Paraphrase Recognition, Textual Entailment is a directional relation because the hypothesis can be inferred from the text but not necessarily the opposite way.

Many NLP tasks like document summarization (SUM),Information Retrieval (IR), Information Extraction (IE) and Question Answering (QA) can take advantage of Paraphrasing and Textual Entailment.

The most common approaches for these tasks and their required resources are:

- **Logic based**: theorem provers, knowledge bases, inference rules and logical conversions.
- **Machine learning based**: annotated examples.
- **Decoding based**: substitution rules, knowledge bases.
- **Semantics based**: semantic networks.
- **Syntactic based**: syntactic parsers.

In this work we experiment with the impact of lexical, syntactic and semantic techniques for Paraphrase Recognition and Textual Entailment.

The *lexical approaches* operate directly with the input strings without making important changes to them; however, sometimes pre-processing is required.

The *syntactic approaches* aim to analyze sentences to show how their words interact with each other. For this approach we need to obtain a syntactic tree that can be obtained by using a syntactic parser.

The *semantic approaches* usually operate on a shallow semantic level; more specifically, for this work we use *Lexical Semantics* which mainly consist of similarity measures between words and semantic relations (hypernym, hyponym, meronymy, holonymy, ..., etc.) obtained from thesauri or semantic networks like WordNet.

We experiment with several combinations of these approaches and auxiliary techniques to find out which combination has the best performance. In the following sections we describe the implemented NLP techniques, and report the results of each experiment that we performed.

2 Implemented NLP techniques

In this section we present the lexical, syntactic and semantic techniques we implemented for Paraphrase Recognition and Textual Entailment.

Particularly we will show examples for Textual Entailment, being **T** the Text and **H** the Hypothesis, although many of these techniques were used for Paraphrasing as well, considering indistinctly one expression of the paraphrasing as Text, and other as Hypothesis. See Section 3 for details on the experiments for Paraphrase Recognition.

2.1 Lexical Module

In this section we describe the Lexical Module we implemented. This module operates at a shallow level of the given texts.

The general algorithm for the lexical module consists in measuring the ratio of coverage of the hypothesis by the given text, this means that, the more words are covered in the hypothesis, the more likely they are to be a textual entailment pair.

We use a coverage threshold **TH** to decide if a given pair is or is not a textual entailment pair; that is, if the coverage ratio of **H** is greater or equal than **TH** the answer will be "YES" otherwise the answer is "NO".

For example, given the pair:

- **T**: The Aztecs were a civilization based on war. Most of them were warriors.
- **H**: The Aztecs were warriors.

we can see that the words of **H** that appear on **T** are "The," "Aztecs," "were" and "warriors." Therefore, the coverage ratio of **H** is $\frac{4}{4} = 1$ so, if **TH** were 0.7 then the answer would be "YES".

The general algorithm for this process is the following:

Given a pair of expressions T, H and a threshold TH:

$H \leftarrow$ preprocessing(H)
$T \leftarrow$ preprocessing(T)
$L_H \leftarrow$ length(H)
$common \leftarrow 0$
for all words *w* in H **do**
  if $contains(T, w)$ **then**
    $common \leftarrow common + 1$
  end if
end for
$coverage \leftarrow common/L_H$
if $coverage \geq TH$ **then**
  return "YES"
else
  return "NO"
end if

The main part of the algorithm is the preprocessing step which receives a text fragment (T or H) and changes it with an auxiliary technique like *stemming, stop words, similarity measures* or *negation detection* in order to obtain a better result. The following sections describe how these techniques are applied.

Preprocessing: Removing Stop Words. Like we mentioned before, stop words are words that in many cases can be removed from a natural language text fragment without loosing critical information, because stop words are very frequent words that appear in most of the text fragments, and therefore the information conveyed by these words usually is not relevant. There is not a definitive list of stop words; however, most common stop word lists consists of *prepositions* and *determiners*.

Our preprocessing system that handles stop words takes as input a text fragment and returns it without the stop words that it contains. For example:

Input: Bountiful arrived after war's end, sailing into San Francisco Bay 21 August 1945. Bountiful was then assigned as hospital ship at Yokosuka, Japan, departing San Francisco 1 November 1945.
Output: Bountiful arrived war's end, sailing San Francisco Bay 21 August 1945. Bountiful was assigned hospital ship Yokosuka, Japan, departing San Francisco 1 November 1945.

The intuition of this approach is that stop words can produce noise affecting the coverage ratio hiding true entailment pairs to our recognition system. This can be seen in the following example:

- **T**: After playing, the dog sat on the mat.
- **H**: A dog sat over a mat.

Here the coverage ratio is $\frac{3}{6} = 0.5$ that could be marked by the system as a *false* textual entailment pair if the threshold was greater than 0.5, but we can see that this is a true textual entailment pair.

Now, lets remove the stop words in the text and the hypothesis:

- **T'**: playing, dog sat mat.
- **H'**: dog sat mat.

where the coverage ratio is $\frac{3}{3} = 1$, showing that this is a *true* textual entailment pair, like it was supposed to be.

Preprocessing: Stemming. The process of stemming consists in deleting the non essential part of the words such as suffixes and prefixes in order to obtain the essential part or *stem* of it. For example in the words **engineer**ing, **engineer**ed and **engineer** the essential part or stem of the words is **engineer**.

Stemming is used to improve retrieval effectiveness because it allows to match words that are not directly identical but their stem is the same. There are several stemming algorithms [15], each one performing the stemming task in different ways, but one of the most popular stemming algorithms for English is Porter's algorithm [18], the one used in this work.

Our preprocessing module that handles stemming takes as an input a natural language text fragment and returns it with the stem representation of each word. For example:

Input: Bountiful arrived after war's end, sailing into San Francisco Bay 21 August 1945. Bountiful was then assigned as hospital ship at Yokosuka, Japan, departing San Francisco 1 November 1945.
Output: Bounti arriv after war' end, sail into San Francisco Bai 21 August 1945. Bounti wa then assign as hospit ship at Yokosuka, Japan, depart San Francisco 1 Novemb 1945.

The intuition of this approach is that we can match words that are not directly the same, but that share the same stem so that they are related in some way and can uncover disguised relationships. Consider the following example:

– **T**: After eating and playing with the kids, the doggy started sleeping.
– **H**: The dog played.

Here the coverage ratio is $\frac{1}{3} = 0.33$, that is a low coverage ratio considering that this is a true textual entailment pair.

Now, lets stem both text and hypothesis:

– **T'**: After eat and plai with the kid, the doggi start sleep.
– **H'**: The dog plai.

that overcomes the previous coverage ratio with $\frac{2}{3} = 0.66$ that is closer to the correct answer.

Preprocessing: Negation Detection. Negation is present in all languages and in most cases statements are affirmative by default. Negation is used to change the polarity of the statements and typically denotes something unusual or exceptions. At first glance, negation seems easy to deal with because the problem can be thought as the task of simply inverting the polarity of the items covered by the scope of negation; however, this is not always the case.

Unlike affirmative statements, negation is marked by words (*not, no, never*) or affixes (*n't,un-*) and also connective adjuncts can be used to negate positive clauses, such as *neither* and *nor*. Another words that indicate negation are *nobody, none, nowhere*, etc.

There are two levels for handling the negation problem: lexical and syntactic.

The lexical approach uses the shallow representation of the sentences to detect negation, according to [3] *not* and *n't* correspond to 79.61% of negative bearing word occurrences based on the WSJ Penn Treebank.

The syntactic approach tries to discover the negative polarity of sentences by looking for patterns of negation based on a syntactic constituents parse tree, but this method requires using a syntactic parser.

In this work we use the detection of negation in a naïve way by searching for the occurrences of the word *not* and words that end with *n't*. Once found, we negate each word prepending **not_** until we reach either a comma or a period.

Our preprocessing module that handles negation takes as input a text fragment and returns the negative representation of it, for example:

Input: The British government did not initially purchase the weapon and civilian sales were modest. However the U.S. Civil War began in 1860 and the governments of both the United States and the Confederacy began purchasing arms in Britain.

Output: The British government did **not_initially not_purchase not_the not_weapon not_and not_civilian not_sales not_were not_modest**. However the U.S. Civil War began in 1860 and the governments of both the United States and the Confederacy began purchasing arms in Britain.

where we add the prefix *not_* to each word that is covered by the scope of the negation.

The intuition of preprocessing negation is that sometimes the system detects false entailment pairs because it does not consider the polarity of the expressions. An example of this is the following:

- **T**: The new car is fast but is not equipped with mp3 player.
- **H**: The new car is equipped with mp3 player.

As we can see in the previous example, the coverage ratio is $\frac{8}{8} = 1$, which means that independently of the selected threshold, the system would incorrectly mark these T and H as a *true* textual entailment pair.

Lets use now the negation detection technique just described, we would have then the following **T'** and **H'**.

- **T'**: The new car is fast but not_is not_equipped not_with not_mp3 not_player.
- **H'**: The new car is equipped with mp3 player.

After the preprocessing, the coverage ratio is $\frac{4}{8} = 0.5$ that helps the system to know that this may not be a true textual entailment pair.

2.2 Syntactic Module

Once we have discussed the lexical module, we proceed to explain the functionality of our syntactic module which operates at a deeper level of the input expressions **T** and **H**; that is, the syntactic level aims to model how the words of a sentence depend on each other. For this purpose, a previous parsing of T and H is required to obtain a *dependency syntactic parse tree* or a *constituent syntactic parse tree*.

The *dependency tree* shows dependencies between the words of an expression. Each edge is labeled with the dependency of the words that it connects, Figure 1 is an example of this kind of tree. The *constituent tree* indicates how the words of a sentence are grouped in lexical constituents that conform the whole expression in a hierarchy, as shown in Figure 2.

For this module we used the *Stanford syntactic parser* [13] which is able to produce both kind of syntactic parse trees. For dependency trees Stanford parser uses the following notation:

Subj(is, Gaspe)
Det(Gaspe, the)
Obj(is, peninsula)
Det(peninsula, a)

where each line represents an edge of the dependency tree. Figure 1 shows the graphic representation of this particular example.

For constituent trees, Stanford parser uses the following notation:

```
(ROOT
  (S
    (NP (DT the) (NNP Gaspe))
    (VP (VBZ is)
      (NP (DT a) (NN peninsula)))))
```

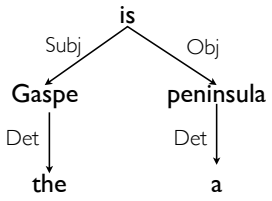

Fig. 1. Example of dependency syntactic parse tree

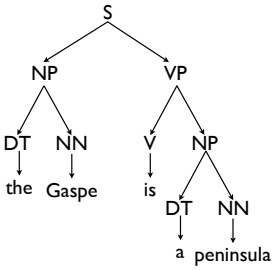

Fig. 2. Example of constituent syntactic parse tree

that corresponds to the tree shown in Figure 2. The general algorithm used in this module measures the coverage ratio of the edges in H's syntactic tree with regard to the syntactic parse tree of T. This means that, the more edges covered in the hypothesis, the more likely will be that it is a textual entailment pair.

In the same way that the lexical approach, we use a coverage threshold **TH** to decide if a given pair is or is not a textual entailment pair.

Both kind of parse trees (dependency and constituent) can be combined as will with complementary techniques like stop words, stemming, and negation, just described in the previous section.

2.3 Syntactic n-grams

Some of the most popular Natural Language Processing techniques are *n*-grams which are sequences of elements as they appear in the texts. The sequence of elements can be composed by words, characters, part of speech tags (POS), etc. The *n* term corresponds to number of elements to be considered by the sequence of elements. For example, the input expression:

– The small funny dog barks,

has the following 2-grams (bigrams): *the small, small funny, funny dog* and *dog barks*, and the following 3-grams (trigrams): *the small funny, small funny dog* and *funny dog barks*, and so on.

In this work we use syntactic n-grams (sn-grams), which are sequences of words that are obtained from the elements appearing in the syntactic trees (dependency or constituent trees) of a sentence. More specifically, sn-grams are constructed by the sequence of nodes that can be reached on any path of length n in the parse tree, namely, this kind of syntactic n-grams is known as continuous syntactic n-grams. In the remainder of this paper we will refer the continuous syntactic n-grams just as syntactic n-grams or sn-grams. There are several types of syntactic n-grams based on the types of elements they take into account:

- **Word sn-grams**: the elements of sn-grams are words.
- **POS sn-grams**: the elements of sn-grams are POS tags.
- **Syntactic relations sn-grams**: the elements of the sn-grams are names of syntactic relations between words.
- **Mixed sn-grams**: they are composed by mixed elements, like words, POS tags and/or syntactic relation types.

The main advantage of sn-grams is that they are based on syntactic relations of words, and thus, each word is bound to its "real" neighbors, ignoring the arbitrariness that is presented in the surface structure [22].

For example, the expression "the small funny dog barks" has the dependency syntactic parse tree illustrated in Figure 3, from which we can obtain the following syntactic bi-grams: *barks dog, dog the, dog funny* and *dog small*, and the following syntactic trigrams: *barks dog the, barks dog funny* and *barks dog small*.

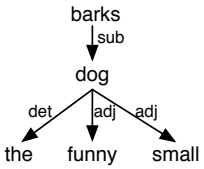

Fig. 3. Example of syntactic n-grams

Syntactic n-grams Extraction In this section we describe the procedure we followed to obtain syntactic n-grams, specifically *s2-grams*, *s3-grams* and *s4-grams* used for our Paraphrase Recognition system.

We based all our procedure on the syntactic dependency trees generated by the Stanford syntactic parser [13].

Our extraction process consists of two main steps:

1. Parse a expression with the Stanford parser.
2. Use the dependency relations obtained from the parse tree to form syntactic 2-grams, s3-grams and s4-grams.

For Step 1, we feed the Stanford parser with an input sentence, for example: "Economic news have little effect on financial markets." then we have as outcome the corresponding syntactic dependency tree, in this example is the following:

nn(news, Economic)
nsubj(have, news)
root(ROOT, have)
amod(effect, little)
dobj(have, effect)
amod(markets, financial)
prep_on(effect, markets)

this observes the notation for syntactic dependency trees defined by the Stanford parser. The above lines are the representation of the tree shown in Figure 4.

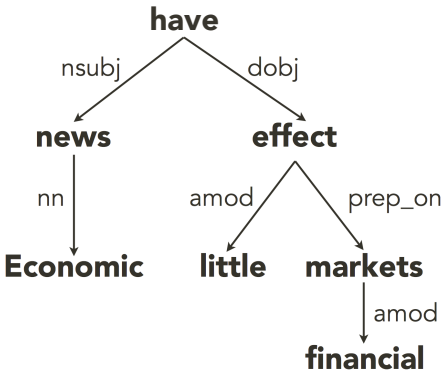

Fig. 4. Dependency tree for the sentence *Economic news have little effect on financial markets*

Syntactic 2-grams Extraction Using the parse tree corresponding to an expression, it is straightforward to obtain the word sn-grams by removing the label of the relation dependency of each edge of the tree, that is:

(news, Economic)
(have, news)

(ROOT, have)
(effect, little)
(have, effect)
(markets, financial)
(effect, markets)

are the s2-grams from the previous example.

Syntactic 3-grams Extraction Once that we have obtained the s2-grams, we can use them to conform the syntactic 3-grams by concatenating two s2-grams "a" and "b" if the second element of "a" is the first element of "b", for example *(have, news)* and *(news, Economic)* conform the s3-gram *(have, news, Economic)*. For the previous example we have the following s3-grams:

(have, news, Economic)
(ROOT, have, news)
(ROOT, have, effect)
(have, effect, markets)
(have, effect, little)
(effect, markets, financial)

Syntactic 4-grams Extraction Based on the syntactic 2-grams and 3-grams, we obtained the syntactic 4-grams. Given a syntactic 3-gram "a" and a syntactic 2-gram "b", we have the syntactic 4-gram "c" if the last element of "a" is the first element of "b", for example for the s3-gram *(have, effect, markets)* and the s2-gram *(markets, financial)* we can conform *(have, effect, markets, financial)* as syntactic 4-gram. For the previous example we have then:

(ROOT, have, news, Economic)
(ROOT, have, effect, little)
(ROOT, have, effect, markets)
(have, effect, markets, financial)

2.4 Semantic Module

A third approach that we experimented with is based on the *semantic level*. The semantics treatment is a very complex task that still has not been completely solved; however, our work has a shallow approach to it.

We are located in *linguistic semantics* that is a subfield of semantics; and linguistics, that studies the meaning of linguistic expressions that not depend on the context. More specifically, we use *Lexical Semantics* that is, the study of relations that exist between words where each one is seen like a node in a graph or a hierarchy, from which is possible to obtain *semantic relations* between them. For example, relations between

hypernyms, hyponyms, synonymy, antonymy, meronymy, and holonymy. In this work we only use the *hypernym* semantic relation and Lin's similarity measure [14] obtained from WordNet.

Preprocessing: Hypernym Representation. Sometimes the direct overlap between words of the given text and hypothesis is not possible; however, it could be the case that two words do not match directly but they match indirectly by means of a semantic relationship like hypernymy, hyponymy, etc. For example, the words *dog* and *canine* are not lexically matched, but using the hypernym relationship we can indirectly associate them, therefore the system would be able to detect "hidden" matches. We based our semantic module in the same algorithm that the lexical module uses; the difference is in the preprocessing step of T and H. This is based on the hypernym semantic relation that takes a text fragment as input and returns the hypernym representation for it. For example:

Input: Female mosquitoes become infected with the malaria parasite when they draw blood from humans with malaria.
Output: Animal dipterous insects become infected with the protozoal infection organism when they gully liquid body substance from homo with protozoal infection.

The intuition for this approach is that a more general representation of the concepts of T and H can be used for looking matches between words with the original T and H fragments, allowing the system to discover matches between words that were hidden before. For example, considering the following input pair:

- **T**: Every one knows that every canine hates felines.
- **H**: Every dog hates cats.

Here the coverage ratio is $\frac{2}{4} = 0.5$, which is marked by the system as a *false* textual entailment pair (the threshold was set to 0.5), but we can see that this is a true textual entailment pair. Now, lets apply the hypernym preprocessing module in the hypothesis:

- **T**: Every one knows that every canine hates felines.
- **H'**: Every canine hates felines.

where the coverage ratio is $\frac{4}{4} = 1$, showing that this is a *true* textual entailment pair, like it was supposed to be.

Semantic Approach with Lin's Similarity Measure. One special situation found in the lexical algorithm described earlier is presented when two words are not equivalent nor they are matched by the hypernym relationship, but they can be matched indirectly. For example, consider the words *spoon* and *fork* that are not directly equivalent but are closely related.

The relatedness treatment of words can help in the Textual Entailment recognition task allowing to recognize hidden matches between words that are not equivalent but are related.

The way on which the similarity or relatedness of words is measured is called *similarity measure*. There are several similarity measures but one of the most used ones is Lin's similarity measure [14] that is defined as:

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

which is based on the information content (IC) of two concepts under the WordNet hierarchy.

The modified recognition algorithm based on the similarity measure is as follows:

Given a pair of expressions T, H a threshold TH and a similarity threshold STH:

$T \leftarrow$ preprocessing(T)
$H \leftarrow$ preprocessing(H)
$L_H \leftarrow$ length(H)
$common \leftarrow 0$
for all word w in H **do**
  for all word m in T **do**
    if $equals(w, m)$ **then**
      $common \leftarrow common + 1$
    else
      if $LinSimilarity(w, m) \geq STH$ **then**
        $common \leftarrow common + 1$
      end if
    end if
  end for
end for
$coverage \leftarrow common/L_H$
if $coverage \geq TH$ **then**
  return "YES"
else
  return "NO"
end if

As can be seen, the similarity measure algorithm is almost the same as the general lexical algorithm, with the difference that if a direct match cannot be found for a given word "m" of the hypothesis, its similarity with each word of the text is calculated, and if it is larger than a specified similarity threshold (*THS*), then the pair is considered as a match.

Additionally we use the combination of the preprocessing modules defined in the lexical section in order to improve the performance of the system.

3 Experiments for Paraphrase Recognition

The development of this section is focused on the Paraphrase Recognition task. The evaluation and tests will be based on the Microsoft Research Paraphrase Corpus (MSRP)

that consists in 5,801 pairs of paraphrase candidates which are divided in two sets: *test* and *training*.

The training set is composed of 4,076 pairs, whereas the test set has 1,725 pairs. Each pair was submitted to human judges who assigned a one label of 1 when it is a true paraphrase pair and 0 otherwise. An example of a pair is the following:

```
1 702876 702977 Amrozi accused his brother, whom he called ``the
witness'', of deliberately distorting his evidence. Referring to
him as only ``the witness'', Amrozi accused his brother of
deliberately distorting his evidence.
```

The Paraphrase Recognition task consists in, given two expressions S_1 and S_2, decide if they are a paraphrase of each other. The following sections describe the process and evaluation of our paraphrase recognition system based on syntactic *n*-grams.

3.1 Recognition Process

Our experiments are based on the idea that syntactic n-grams can provide more information than classic *n*-grams because they work at a deeper level considering the relations and dependencies between words of the input expression.

Syntactic *n*-grams can also be useful for detecting the "real" neighbors of each word ignoring the arbitrariness that is presented on the surface structure such as adjectives before nouns. For example in the previous sentence "the small funny dog barks," we can do the following comparison:

Table 1. n-grams vs. syntactic n-grams.

2-grams	**Syntactic 2-grams**
the small	barks dog
small funny	dog the
funny dog	dog funny
dog barks	dog small

As can be seen in Table 1, traditional 2-grams produce pairs that convey less information like "small funny" or "the small", whereas the syntactic 2-grams are all meaningful pairs.

We experimented with five different approaches for the recognition process:

1. Overlapping Syntactic n-grams.
2. Overlapping Syntactic n-grams and stemming.
3. Overlapping Syntactic n-grams and synonym detection.
4. Overlapping Syntactic n-grams and Lin's similarity measure.
5. Overlapping Syntactic n-grams and linear interpolation.

As in Textual Entailment, the general process for the Paraphrase Recognition system independently of the implemented approach, consists in measuring the differences between the sn-grams of the input expressions $\mathbf{S}_1$ and $\mathbf{S}_2$. A difference threshold **T** is used to decide if the input expressions are or not paraphrases of each other. That is, if the difference between them is less than the threshold T, they are paraphrases; otherwise they are not paraphrases. The algorithm is presenting below:

Given two sentences S_1 and S_2 and a threshold of difference **T**:

$L_1 \leftarrow$ sn-grams(S_1)
$L_2 \leftarrow$ sn-grams(S_2)
if $size(L_1) \geq size(L_2)$ **then**
  $T' \leftarrow size(L_1) \times T$
else
  $T' \leftarrow size(L_2) \times T$
end if
$D \leftarrow$ sn-gramsDifferences(L_1, L_2)
if $D < T'$ **then**
  return "YES"
else
  return "NO"
end if

In the next subsections we describe each of the approaches used for our recognition system, along with the result obtained with each one of them.

We use the *Microsoft Research Paraphrase Corpus* to evaluate the accuracy, precision, recall and F-measure scores of our system. With these values is possible to compare the performance achieved by our Paraphrase Recognition system with other works of the state of the art.

The Microsoft Research Paraphrase Corpus provides two datasets: *train* and *test*. With the train set we adjust the optimal value for our difference threshold to be the one that produces the best scores. Once that we decided the optimal threshold, we use it to evaluate the system with the test set. According to our experiments, the best threshold for our system is **0.85**.

3.2 Overlapping Syntactic N-grams

The simplest approach implemented in our system, is just to count the overlapping syntactic n-grams in the two input expressions S_1 and S_2. We experimented with s2-grams, s3-grams and s4-grams independently.

We repeated each experiment adjusting the maximum difference threshold in the range of 0.2 to 0.9. This approach is based on the general process algorithm. For instance, lets suppose that we are given the following input expressions under a threshold **T** of 0.3:

S_1: A mathematician solved the problem.

S_2: The problem was solved by a mathematician.

applying our s2-grams extraction module we obtain the s2-grams shown on Table 2 for S_1 and S_2 respectively.

Table 2. Syntactic 2-grams corresponding to S_1 and S_2.

s2-grams for S_1	**s2-grams for S_2**
(mathematician, a)	(problem, the)
(solved, mathematician)	(solved, problem)
(ROOT, solved)	(solved, was)
(problem, the)	(ROOT, solved)
(solved, problem)	(mathematician, a)
	(solved, mathematician)

Results. For this example, we can see that T' can be computed from Table 2: **T'** = size(L_2) $\times$ T = 6 $\times$ 0.3 = **2**. The remaining process consist on determining if S_1 and S_2 do not differ by more than 2 words. It can be seen that S_1 and S_2 differ only on the s2-gram *(solved, was)*, therefore this would be considered as a *true* paraphrase pair by the system. The same procedure is applied individually for s3-grams and s4-grams.

Table 3 shows results obtained when we use the basic technique of common syntactic n-grams for syntactic 2-grams, 3-grams and 4-grams.

As we can see in the table, our system achieves the highest F-measure of 80.3% when*s2-grams* are used. However it can also be seen that the higher order of the syntactic n-grams, the higher precision value is obtained. On the other hand, the higher order of syntactic n-grams, the less recall value is obtained.

Table 3. Results of syntactic n-grams approach

	Accuracy	**Precision**	**Recall**	**F-measure**
S2-grams	**68.3%**	**68.3%**	**97.4%**	**80.3%**
S3-grams	64.2%	72.8%	73.6%	73.2%
S4-grams	56.9%	74.4%	53.7%	62.4%

3.3 Overlapping Syntactic N-grams and Stemming

As explained in Section 2.1 for Textual Entailment, Stemming is a process that removes the non essential part of the words such as suffixes and prefixes in order to obtain the *essential* part or stem of a word. For example in the words **fish**ing, **fish**ed, **fish**er the stem is **fish**.

We use the Porter stemming algorithm [18], since it is the most popular stemming algorithm for English, but there are many others.

Our sn-grams post processing module that works with stemming, takes as input a syntactic n-gram and returns the stemmed representation of it, for example:

- *(redness, car)* $\rightarrow$ *(red, car)*
- *(fully, car, engineered)* $\rightarrow$ *(fulli, car, engin)*

The Paraphrase Recognition process based on stemming performs the following steps:

1. Obtain sn-grams for each input expression.
2. Apply stemming for each sn-gram obtain in the previous step.
3. Use the general algorithm.

With this approach we intent to recognize syntactic n-grams that can not be related in a direct way but that may be indirectly related by means of their stems.

Results of Overlapping Syntactic N-grams and Stemming. Now we show in Table 4 the achieved results for the system when post processing of the syntactic n-grams is used, by transforming each one on its stemmed representation.

As the table shows, the stemming of syntactic n-grams slightly increases the general performance of the system, yielding an increase in accuracy, precision, recall and the F-measure.

The F-measure obtained with this approach is 80.6% by using *s2-grams*. Again the precision increases and the recall decreases when a higher degree of syntactic n-gram is used.

Table 4. Results of syntactic n-grams with stemming approach

	Accuracy	**Precision**	**Recall**	**F-measure**
S2-grams	**68.6%**	**68.4%**	**97.9%**	**80.6%**
S3-grams	64.5%	72.5%	74.9%	73.7%
S4-grams	57.8%	74.4%	55.7%	63.7%

3.4 Overlapping Syntactic N-grams and Synonym Detection

One way of finding "hidden" related pairs of words in sentences or related pairs of syntactic n-grams in the paraphrase recognition process is by using synonyms. For instance, suppose the syntactic 2-grams: **(car, red)** and **(automobile, carmine)**: our basic paraphrase recognition system would not match this pair, but it can be seen that they should be considered as a match.

In order to cope with this kind of problems, we introduce a synonym detection module based on WordNet synsets that are considered as groups of synonym words.

Two words are synonyms if they can be interchanged under the same scope without modifying the truth value of an expression, in other words, for our purpose, synonyms are words that are considered equivalent. For example, the words *plant* and *flower*, *home* and *house*, *kid* and *child* can be considered as equivalent.

Applying this intuition to our Paraphrase Recognition system, a post processing module was developed. This module takes a syntactic n-gram as input and returns four equivalent syntactic sn-grams by obtaining the most common synonym of each word from WordNet.

Taking the first example of this section, if we apply our module to *(car, red)*, we obtain the following:

- (car, red)
- (auto, red)
- (car, redness)
- (auto, redness)

This answer is conformed by the combination of the synonyms of the words *car* and *red*, the combination is required because we need to compare each possible variation of the pair, augmenting the recognition capability of the system.

The result of applying the same procedure to *(automobile, carmine)* is:

- (automobile, carmine)
- (auto, carmine)
- (automobile, red)
- (auto, red)

From the previous two answers the system now is able to detect the match between *(car, red)* and *(automobile, carmine)* because both share the same synonym form *(auto, red)*.

A similar process is used for syntactic 3-grams and 4-grams treatment, the only difference is that we obtain 9 and 16 synonym combinations respectively.

Results of Overlapping Syntactic N-grams and Synonyms. Table 5 shows the system performance when we apply synonym postprocessing of sn-grams. This time the synonym approach helps to improve the basic approach (direct overlapping), but it does not contribute as much as stemming.

The highest F-measure is 80.4% with *s2-grams* again.

Table 5. Results for the synonym approach

	Accuracy	Precision	Recall	F-measure
S2-grams	**68.4%**	**68.3%**	**97.7%**	**80.4%**
S3-grams	66.4%	66.9%	97.6%	79.4%
S4-grams	65.4%	67%	94.5%	78.4%

3.5 Overlapping Syntactic N-grams and Lin's Similarity Measure

Another related problem found on the process of recognizing a match between syntactic n-grams is presented when two elements are not directly equivalent and also not matched by using the synonym technique described in the previous section. An example is the following pair of syntactic 2-grams: **(song, romance)** and **(music, love)**. In this case there is not a direct match between words and, if we consider the synonyms shown on the Table 6, we can see that no match can be found. For this reason we use

Table 6. Example: Synonym approach.

Synonyms (song, romance)	**Synonyms (music, love)**
(song, romance)	(music, love)
(vocal, romance)	(euphony, love)
(song, romanticism)	(music, passion)
(vocal, romanticism)	(euphony, passion)

an additional approach based on word similarity measures. More specifically, we use the same similarity measure we used for Textual Entailment shown in Section 2.4. For reader's convenience, we reproduce here the formula for its calculation: [14]:

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

Using this measure we try to detect similar syntactic n-grams by considering two corresponding words as a match if their similarity measure is greater than, or equal to, a threshold value.

The Lin similarity measure between *song* and *music* is 0.86 and between *romance* and *love* is 0.53; thus, if the threshold of similarity were 0.5, the pair of syntactic 2-grams (*song, music*) and (*romance, love*) would be considered as a match.

Overlapping Syntactic N-grams and Lin Similarity. Now we present the results obtained by the system when we use a comparison technique between sn-grams with Lin's similarity measure, instead of directly comparing them.

Table 7 shows that the performance of the system decreases in accuracy, precision, recall and F-measures with regard to the previous shown approaches. The highest F-measure obtained is 80.1%, again with the syntactic *s2-grams*.

3.6 Overlapping Syntactic N-grams and Linear Interpolation

An additional approach that we tried in order to improve the system performance is based on the linear interpolation technique used in traditional n-grams that consists in creating a linear interpolation of the syntactic 4-grams, 3-grams and 2-grams models, each one weighted by a λ value, where the sum of all lambdas must be equal to 1.

Table 7. Results of syntactic n-grams with Lin's similarity

	Accuracy	Precision	Recall	F-measure
S2-grams	**67.4%**	**67.4%**	**98.7%**	**80.1%**
S3-grams	61.1%	72.1%	77.3%	74.6%
S4-grams	57.3%	73.3%	56.4%	63.8%

We think that overlapping s4-grams mean a stronger relationship between two sentences compared with s3-grams. We considered the same idea between s3-grams and s2-grams, therefore, we assigned greatest weight to s4-grams, then a smaller weight for s3-grams and the lowest weight was assigned to s2-grams. The intuition here is that the higher the degree of overlapping n-grams, the more similar two texts are.

We represent this idea as follows:

$$F_S = \lambda_4 \times snGrams(S_1, S_2, 4) + \lambda_3 \times snGrams(S_1, S_2, 3) + \lambda_2 \times snGrams(S_1, S_2, 2)$$

Where snGrams(S_1, S_2, n) represents the common syntactic n-grams of degree *n* between the expressions S_1 and S_2 and $\lambda_4 + \lambda_3 + \lambda_2 = 1$.

Consider the following example:

- $\mathbf{S}_1$: A mathematician solved the problem.
- $\mathbf{S}_2$: The problem was solved by a mathematician.

As we can see from the Table 8 the two input expressions do not have overlapping syntactic 4-grams, therefore our system based on s4-grams would mark the expressions as a *false* paraphrase pair; however, it can be seen that by applying linear interpolation we are able to obtain an score grater than zero, allowing the system to consider a different answer. The computation using this approach is:

$$F_S = 0.5 \times 0 + 0.3 \times 2 + 0.2 \times 4 = 1.4,$$

where $\lambda_4 = 0.5, \lambda_3 = 0.3, \lambda_2 = 0.2$.

The obtained value is then normalized to get a final score between 0 and 1 that can be used together with a similarity threshold to decide if the expressions are or are not a true paraphrase pair.

Results of Overlapping Syntactic N-grams and Linear Interpolation. Now we present in Table 9 the results obtained for the system when we use the linear interpolation approach. As can be seen, the system performance decreases considerably when using Linear Interpolation; in part, due to the fact that the lambda values require an exhaustive procedure to obtain an optimal value for them. We experimented with several lambda values with low success.

Table 8. Linear interpolation of syntactic n-grams

n	**Input 1**	**Input 2**	**snGrams(S_1, S_2, n)**
2	(mathematician, a) (problem, the) (solved, problem) (solved, mathematician)	(mathematician, a) (problem, the) (solved, problem) (solved, mathematician) (solved, was)	4
3	(solved, mathematician, a) (solved, problem, the)	(solved, mathematician, a) (solved, problem, the)	2
4	No	No	0

For this particular approach we used a threshold of 0.2. Note that in this case the algorithm is based on similarity, contrasting with the other approaches that are based on distance.

Table 9. Results with linear interpolation

$\lambda_4, \lambda_3, \lambda_2$	**Accuracy**	**Precision**	**Recall**	**F-measure**
0.5, 0.3, 0.2	60.1%	75.1%	59.8%	66.6%
0.4, 0.3, 0.3	**62.9%**	**74.6%**	**67%**	**70.6%**
0.7, 0.2, 0.1	57.8%	76.6%	52.6%	62.4%

The best F-measure shown in Table 9 is 70.6%, with $\lambda_4 = 0.4, \lambda_3 = 0.3$ and $\lambda_2 = 0.3$.

4 Experiments for Textual Entailment

In this section we describe and compare the experiments for the different approaches described in Section 2 and their combinations used by our system. The evaluation of this system is based on the PASCAL RTE-3 dataset. We used the development set for calibrating the best threshold value and then we used the test set for the final evaluation. The system evaluation is based on the accuracy, precision, recall and the F-measure score; however we focus mainly on the *accuracy* measure, because that is the one used in the RTE challenge to compare results.

4.1 The Lexical Approach Results

The Table 10 shows the scores obtained by our system with the lexical approach, it shows the basic approach (simple overlapping) and all the combinations of complementary techniques used. Notice how the *stemming* and *removing stop words* techniques contribute in both ways, individually and together to improve the performance of the base system that does not use any additional techniques.

An important observation is that apparently the negation treatment not only does not help to improve the results but also affects it.

We conclude that the best combination under the auxiliary techniques that we used for the lexical module is using *stemming* and *removing stop words* together, reaching and accuracy of 66% and precision of 61.5%, with a threshold of 0.5.

Table 10. Lexical results

Stemming	Stop words	Negation	Accuracy	Precision	Recall	F-measure	Threshold
			64.3%	62.8%	74.3%	68.1%	0.65
X			64.7%	61.5%	82.9%	70.6%	0.65
	X		65.2%	62.2%	81.9%	70.7%	0.55
		X	61.1%	61.4%	64.6%	63.0%	0.50
X	**X**		**66.0%**	**61.5%**	**89.7%**	**73.0%**	**0.50**
	X	X	65.0%	62.0%	81.4%	70.4%	0.55
X		X	64.2%	62.9%	73.4%	67.7%	0.70
X	X	X	65.6%	61.2%	89.5%	72.7%	0.50

4.2 The Syntactic Approach Results

Now we discuss the syntactic approach results. Our intuition was that the deeper the level of analysis, the more accurate is the system; however, by using a basic syntactic approach, we got lower results; maybe because a more sophisticated syntactic technique is required.

Table 11 shows the achieved results. For each experiment we determined the best threshold. We first tried a simple edge overlapping technique; then we tried splitting both T and H by periods with the idea of creating a more accurate parsing. We also present the results of using constituent trees and the auxiliary techniques applied after the parsing step.

The best score achieved is an accuracy of 58.5% and precision of 56.8% under a threshold of 0.1.

Table 11. Syntactic results

Overlapping Edges	Accuracy	Precision	Recall	F-measure	Threshold
Simple	57.0%	56.4%	70.0%	62.5%	0.05
Splitting by periods	57.8%	56.8%	73.6%	64.1%	0.05
Constituents	54.2%	52.9%	95.1%	68.0%	0.30
Stop words	57.3%	59.0%	55.1%	56.9%	0.15
Negation	56.8%	57.0%	64.1%	60.3%	0.05
Stemming	**58.5%**	**56.8%**	**79.2%**	**66.1%**	**0.10**

4.3 The Semantic Approach Results

Table 12 shows the results achieved by our semantic approach. Conversely to the lexical approach, negation treatment gives some benefit, but not enough to reach a higher score. Note also that in this case, word similarity does not contribute to overcome the results. The best combination was *hypernym, stemming* and *stop words* altogether, with an accuracy of 64.0% and a precision of 61.0% with a threshold of 0.65.

Table 12. Semantic results

Technique	Accuracy	Precision	Recall	F-measure	Threshold
Hypernym	62.6%	63.7%	62.6%	63.2%	0.70
Hypernym and stemming	62.2%	59.6%	81.2%	68.8%	0.60
Hypernym and stop words	62.6%	60.1%	80.4%	68.8%	0.55
Hypernym, stop words and word sim.	60.3%	58.2%	80.0%	67.4%	0.80
Word similarity	55.3%	53.6%	94.1%	68.3%	0.10
Word similarity and stemming	54.3%	53.0%	96.0%	68.3%	0.10
Word similarity and stop words	60.0%	60.6%	62.6%	61.6%	0.25
Word similarity, stop words and stemming	60.5%	60.5%	65.6%	62.9%	0.25
Hypernym, stop words and negation	62.5%	60.1%	79.7%	68.5%	0.55
Hypernym and negation	63.5%	61.0%	79.5%	69.0%	0.65
Hypernym, stemming and stop words	**64.0%**	**61.0%**	**81.9%**	**70.0%**	**0.65**

5 Conclusions and Future Work

5.1 Paraphrase Recognition

Summarizing the result tables shown in Section 3, it can be seen that the best scores the system yields is an F-measure of **80.6%**, this is obtained by using the auxiliary post processing technique of stemming, applied to each syntactic n-gram obtained in the syntactic parse step.

The synonym technique is also a good approach since the performance of the system improved with regard to the basic algorithm, which consists on simple syntactic n-grams overlapping; however, it does not contribute as much as the stemming technique.

On the other hand the Lin's similarity measure and linear interpolation approaches that we applied do not seem to contribute for this specific task.

After experimenting with syntactic n-grams and some complementary techniques, we conclude that syntactic n-grams can be used successfully achieving good results in the Paraphrase Recognition task; however there is still a lot of room for improvement.

Table 14 shows the unsupervised reported scores using the Microsoft Research Paraphrase Corpus and we also show our results in order to compare them with these works.

As future work, a deeper analysis can be done, dealing on how to apply similarity measures to compare syntactic n-grams and also how to choose the optimal *lambda*

values for the linear interpolation approach. Another approach that remains to be tested is the use of non-continuous syntactic n-grams [21] since the syntactic n-grams used in this work were continuous syntactic n-grams.

Table 13. Unsupervised reported works

Author	Accuracy	F-measure
Fernando and Stevenson, 2008	74.1%	82.4%
Islam and Inkpen, 2007	72.6%	81.3%
Mihalcea et al., 2006	70.3%	81.3%
our system, 2013	**68.6%**	**80.6%**
Rus et al., 2008	70.6%	80.5%
Mihalcea et al., 2006	65.4%	75.3%

5.2 Conclusions for Textual Entailment

Comparing Tables 10, 11 and 12, we can see that the best result achieved by our system was obtained under the simple lexical approach with an accuracy of 66% and a precision of 61.5%. Although this is not the highest score compared with the state of the art for RTE-3, it shows that by using a few additional resources it is possible to obtain fair results. This is good news, because in some languages like *Tagalog*, complex resources could represent a restriction for Textual Entailment recognition systems.

In Table 14 we can see the top 8 reported results on RTE-3 compared with our system, showing that most of them use several auxiliary resources like theorem provers, knowledge bases, logical inference and so on.

Table 14. Resources used

Author	Accuracy	Precision	Lexical	Syntactic	Semantic	Logical Inference	Background knowledge	Machine Learning
Hickl	80%	88.15%	X		X	X	X	X
Tatu	72.25%	69.42%	X		X	X	X	
Iftene	69.1%	-	X	X			X	
Adams	67%	-	X				X	X
Zanzotto	66.7%	66.7%	X	X			X	
Wang	66.5%	-		X				X
Us	**66%**	**61.5%**	**X**					
Blake	65.8%	60.96%	X	X				X
Ferrandez	65.6%	-	X	X				

As future work, negation treatment can be improved, as well as the syntactic technique used for this work, taking advantage of more information Linguistics can provide.

References

1. Androutsopoulos, I. & Malakasiotis, P. (2010). "A Survey of Paraphrasing and Textual Entailment Methods. *Journal of Artificial Intelligence Research*, 38 135-187.
2. Bar-Haim, R., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B. & Szpektor, I. (2006). "The Second PASCAL Recognising Textual Entailment Challenge", *Proceedings of the Second PASCAL Challenges Workshop on Recognizing Textual Entailment*, Venice, Italy.
3. Blanco, E. & Moldovan, D. (2011). "Some Issues on Detecting Negation from Text", *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference*, Florida, Association for the Advancedment of Artificial Intelligence, 228-233.
4. Bos, J. & Markert, K. (2005). "Recognising Textual Entailment with Logical Inference", *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, Vancouver, October 2005. Stroudsburg, PA, USA: Association for Computational Linguistics, 628-635.
5. Dagan, I., Glickman, O. & Magnini, B. (2006). The PASCAL Recognising Textual Entailment Challenge. In: Quiñonero-Candela, J., Dagan, I., Magnini, B. & D'Alchébuc, F. eds. (2006). *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg, pp. 177-190.
6. Das, D. & Smith, N. (2009). "Paraphrase Identification As Probabilistic Quasi-synchronous Recognition", Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP, Suntec, Singapore, August. Stroudsburg, PA, USA: Association for Computational Linguistics, 468-476.
7. Dolan, 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*, Geneva, Switzerland, Stroudsburg, PA, USA: Association for Computational Linguistics.
8. Dolan, W. & Brockett, C. (2005). "Automatically Constructing a Corpus of Sentential Paraphrases", *Proceedings of the 3rd International Workshop on Paraphrasing*, Jeju island, Korea, 9-16.
9. Giampiccolo, D., Magnini, B., Dagan, I. & Dolan, B. (2007). "The Third PASCAL Recognizing Textual Entailment Challenge", *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, Prague, Czech Republic, Stroudsburg, PA, USA: Association for Computational Linguistics, 1-9.
10. Hickl, A. & Bensley, J. (2007). "A Discourse Commitment-based Framework for Recognizing Textual Entailment", *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, Prague, Czech Republic, Stroudsburg, PA, USA: Association for Computational Linguistics, 171-176.
11. Jurafsky, D. & Martin, J. (2008). *Speech and language processing*, (2nd edition). Upper Saddle River, N.J.: Prentice Hall.
12. Kauchak, D. & Barzilay, R. (2006). "Paraphrasing for Automatic Evaluation", *Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*, New York, June 2006. Association for Computational Linguistics, 455-462.
13. Klein, D. & Manning, C. (2003). "Accurate Unlexicalized Parsing", *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, Sapporo, Japan, Stroudsburg, PA, USA: Association for Computational Linguistics, 423-430.

14. Lin, D. (1998b). "An Information-Theoretic Definition of Similarity", *Proceedings of the Fifteenth International Conference on Machine Learning*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 296-304.
15. Manning, C., Raghavan, P. & Schtze, H. (2008). *Introduction to information retrieval*. New York: Cambridge University Press.
16. Mihalcea, R., Corley, C. & Strapparava, C. (2006). "Corpus-based and Knowledge-based Measures of Text Semantic Similarity", *Proceedings of the 21st National Conference on Artificial Intelligence*, Boston, Massachusetts, American Association for Artificial Intelligence, 775-780.
17. Miller, G. (1995). WordNet: A Lexical Database for English. *Communications of the ACM*, Iss. Nov. 1995 39-41.
18. Porter, M. (1997). An algorithm for suffix stripping. *Readings of Information Retrieval*, 313-316.
19. Qiu, L., Kan, M. & Chua, T. (2006). "Paraphrase Recognition via Dissimilarity Significance Classification", *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, Sydney, Australia, Stroudsburg, PA, USA: Association for Computational Linguistics, 18-26.
20. Ros Gaona, M., Gelbukh, A. & Bandyopadhyay, S. (2010). "Recognizing Textual Entailment Using a Machine Learning Approach", *Proceedings of the 9th Mexican International Conference on Artificial Intelligence Conference on Advances in Soft Computing: Part II*, Pachuca, Mexico, Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 177-185.
21. Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A. & Chanona-Hernández, L. (2013). "Syntactic Dependency-based N-grams: More Evidence of Usefulness in Classification", *Conference on Intelligent Text Processing and Computational Linguistics*, Springer-Verlag Berlin Heidelberg, 13-24.
22. Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A. & Chanona-Hernández, L. (2012). Syntactic Dependency-based N-grams as Classification Features. In: Batyrshin, I. & González, M. eds. (2013). *Advances in Computational Intelligence*. Springer-Verlag Berlin Heidelberg, pp. 1-11.
23. Tatu, M. & Moldovan, D. (2006). "A Logic-based Semantic Approach to Recognizing Textual Entailment", *Proceedings of the COLING/ACL on Main Conference Poster Sessions*, Sydney, Australia, Stroudsburg, PA, USA: Association for Computational Linguistics, 819-826.
24. Tatu, M. & Moldovan, D. (2005). "A Semantic Approach to Recognizing Textual Entailment", *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP)*, Vancouver, British Columbia, Canada, Stroudsburg, PA, USA: Association for Computational Linguistics, 371-378.
25. Zanzotto, F., Pennacchiotti, M., & Moschitti, A. (2009). "A machine learning approach to textual entailment recognition." *Natural Language Engineering*, 15 (04), 551-582.