IT3105 Project III: Recognizing Textual Entailment Project Description

Erwin Marsi

Fall 2011
Version 1.0

Overview

The goal of this project is to implement various systems for *Recognizing Textual Entailment* (RTE). Textual entailment is a relation between two text fragments, a text T and a hypothesis H. We say that a text T *entails* a hypothesis H if, typically, a human who reads T would infer that H is most likely true. Some examples of textual entailment, both true and false, are given in Table 1.

The assignment is divided into four major parts, each scored individually as indicated in the description of each task:

Part I: Lexical matching (25 points)
Implement simple, knowledge-poor RTE systems based on string matching techniques.

Part II: Syntactic matching (25 points)
Implement an RTE system exploiting syntactic analysis through tree edit distance calculation.

Part III: Machine learning (25 points)
Implement an RTE system based on general machine learning techniques.

Part IV: Your very own RTE system (25 points)
Extend and optimize any of the RTE systems from the previous parts in order to get best performance

A detailed description of each part – as well as the scores that can be obtained for completing subparts of each part – is provided in Sections below. These descriptions use some concepts and terminology that you are probably not familiar with. To have sufficient background to understand

Table 1: Examples of textual entailment

Id	Text	Hypothesis	Entails
1	The drugs that slow down or halt Alzheimer's disease work best the earlier you administer them.	Alzheimer's disease is treated using drugs.	YES
2	Drew Walker, NHS Tayside's public health director, said: "It is important to stress that this is not a confirmed case of rabies."	A case of rabies was confirmed.	NO
3	Yoko Ono unveiled a bronze statue of her late husband, John Lennon, to complete the official renaming of England's Liverpool Airport as Liverpool John Lennon Airport.	Yoko Ono is John Lennon's widow.	YES
4	Arabic, for example, is used densely across North Africa and from the Eastern Mediterranean to the Philippines, as the key language of the Arab world and the primary vehicle of Islam.	Arabic is the primary language of the Philippines.	NO
5	About two weeks before the trial started, I was in Shapiro's office in Century City.	Shapiro works in Century City.	YES
6	Meanwhile, in his first interview to a Western print publication since his election as president of Iran earlier this year, Ahmadinejad attacked the "threat" to bring the issue of Iran's nuclear activity to the UN Security Council by the US, France, Britain and Germany.	Ahmadinejad is a citizen of Iran.	YES

this assignment, you are therefore advised to read the lecture notes entitled `project-rte-lecture-notes.pdf` and to attend the accompanying lectures. In any case, make sure to carefully read the lecture notes as well as the full project description *before* you start coding. If in doubt, contact the instructors with your questions.

You are free to use any computer language for this assignment. Some supporting Python code for reading input files and scoring output files is provided on the project website, if you choose to use Python or want to use our code as pseudocode for translation to a different language. Whatever your weapon of choice may be, make sure that it has decent support for processing/parsing XML.

The project can be done alone or in groups of two students, but no more than two.

The following Sections first describe the Recognizing Textual Entailment task in more detail, then the four major parts of the assignment, and finally the deliverables expected at the end of the project.

The Recognizing textual Entailment challenge

The Recognizing Textual Entailment (RTE) challenge is a competition (many researchers prefer the more friendly term *shared task*) that, starting from 2004, has been held yearly. Even though the exact task has varied over time, the common theme is that participants need to build a system that can recognize textual entailments. This project will be based on the RTE-2 challenge from 2005-2006.

Task definition

We consider an applied notion of textual entailment, defined as a directional relation between two text fragments, termed T - the entailing *text*, and H - the entailed *hypothesis*. We say that T entails H if, typically, a human reading T would infer that H is most likely true. This somewhat informal definition is based on (and assumes) common human understanding of language as well as common background knowledge. Textual entailment recognition is the task of deciding, given T and H, whether T entails H. Some examples of textual entailment are given in Table 1.

Some additional judgment criteria and guidelines are listed below (examples are taken from Table 1):

- Entailment is a directional relation. The hypothesis must be entailed from the given text, but the text need not be entailed from the hypothesis.

- The hypothesis must be fully entailed by the text. Judgment would be NO if the hypothesis includes parts that cannot be inferred from the text.
- Cases in which inference is very probable (but not completely certain) are judged as YES. In example 5 one could claim that although Shapiro's office is in Century City, he actually never arrives at his office, and works elsewhere. However, this interpretation of T is very unlikely, and so the entailment holds with high probability. On the other hand, annotators were guided to avoid vague examples for which inference has some positive probability which is not clearly very high.
- Our definition of entailment allows presupposition of common knowledge, such as: a company has a CEO, a CEO is an employee of the company, an employee is a person, etc. For instance, in example 6, the entailment depends on knowing that the president of a country is also a citizen of that country.

Input data format

Data sets are formatted as XML files, containing a number of T-H pairs in the following format:

```
<pair id="id_num" entailment="YES|NO" task="task_acronym">

   <t> the text... </t>

   <h> the hypothesis... </h>

</pair>
```

where:

- each T-H pair appears within a single `<pair>` element.
- the element `<pair>` has the following attributes:
  - `id`, a unique numeral identifier of the T-H pair.
  - `task`, the acronym of the application setting from which the pair has been generated: `IR`, `IE`, `QA` or `SUM` – this aspect is irrelevant for the current project
  - `entailment` (in the development set only), the gold standard entailment annotation, being either `YES` or `NO`.
- the element `<t>` (text) has no attributes, and it may be made up of one or more sentences.

- the element `<h>` (hypothesis) has no attributes, and it usually contains one simple sentence.

The data is split into a *development set* and a *test set*, to be released separately. The goal of the development set is to guide the development and tuning of your systems. It contains 800 pairs of text and hypothesis which are labeled for textual entailment. The dataset is balanced: 400 pairs in which the entailment relation holds and 400 pairs in which the entailment relation does not hold. This means that random guessing will (on average) give you accuracy score of 50%. The objective of your system is of course to get an accuracy score as close to 100% as possible.

The test set will be supplied at the end of the project. It has exactly the same format as the development data, except that it is blind, i.e. the entailment attribute is missing.

Preprocessed input data

In addition, a preprocessed version of the same data is supplied. The preprocessing includes sentence splitting and syntactic parsing using the MINIPAR dependency parser; see the lecture notes for more information on linguistic aspects including parsing. Note that since the preprocessing is done automatically, it does contain some errors.

The preprocessed format follows the XML structure described earlier, but instead of text/hypothesis sentences, it contains syntactically analysed versions of these sentences. The hypothesis usually consists of a single sentence, whereas the text often consists of multiple sentences. An example of the analysis for the hypothesis sentence "*Wilson was awarded the Pulitzer Prize.*" is given in Figures 1 and 2.

The analysis takes the form of a list of nodes generated by the MINIPAR parser. There are two node types: those that describe an actual word in the sentence (see Figure 1) and "artificial" nodes (see Figures 2). They can be differentiated by their ID. Nodes describing a word contain a numeric id. Artificial node's id start with 'E' and then a number.

Each node that describes a word contains the word surface form, its lemma and part-of-speech (POS) tag. For example, node 3 has the surface form *awarded*, the lemma *award* (the underlying form shared by all surface variants *award*, *awards*, *awarded* and *awarding*), and POS tag *V* (since it is a Verb). Artificial nodes may also contain a lemma and a POS tag. All information from the XML structure is represented in a more convenient format in Table 2. Notice that it is not required for the project to really understand what all the different POS tags and dependency relations mean. However, if you are interested in the details, see the document `minipar-format.pdf` available from the project website.

Each node (with the exception of a root node) contains a pointer to

Table 2: Sample of preprocessed data in table format

id	word	lemma	pos	parent	relation
1	Wilson	Wilson	N	3	s
2	was	be	be	3	be
3	awarded	award	V	E0	i
4	the	the	De	6	lex-mod
5	Pulitzer	Pulitzer	U	6	lex-mod
6	Prize	Pulitzer Prize	N	3	obj2
7	.	.	U	E1	punc
E0		fin	C		
E1			U		
E2		Wilson	N	3	obj1

the id of its parent node and the dependency relation with this parent. For example, node 3 has parent node E0 and the dependency relation is *i* (indicating that *awarded* is a predicate of an inflected clause). These pointers from nodes to their parent define a dependency structure or dependency graph. The dependency structure for our example is shown in Figure 3. This will usually be a tree, where the root of the tree is simply the node that has no parent. Sometimes the dependency structure takes the form of a forest of two or more unconnected trees, due to parsing errors. This is the case in our example, because the tree rooted in node E1 is not connected to the main tree rooted in node E0.

Output data format

Output produced by RTE systems must conform to the RTE output format. This is a file starting with the line `ranked:<blank space>no`. Furthermore it must have one line for each T-H pair in the data set, in the format `pair_id<blank space>judgment` where

- `pair_id` is the unique identifier of each T-H pair, as it appears in the data set.
- judgment is either `YES` or `NO`.

For example, suppose that the pair identifiers in the test set are 1...6. A valid output file may look as follows:

```
ranked: no
1 NO
2 NO
3 YES
```

```
4 YES
5 NO
6 YES
```

Evaluation

The judgments (classifications) returned by the system are compared to those manually assigned by the human annotators (the Gold Standard). The percentage of matching judgments will provide the *accuracy* of the run, i.e. the fraction of correct responses. A Python script for evaluation called `eval_rte.py` is available from the project website.

```
<hypothesis>
    <sentence serial="1">
        <node id="1">
            <word>Wilson</word>
            <lemma>Wilson</lemma>
            <pos-tag>N</pos-tag>
            <relation parent="3">s</relation>
        </node>
        <node id="2">
            <word>was</word>
            <lemma>be</lemma>
            <pos-tag>be</pos-tag>
            <relation parent="3">be</relation>
        </node>
        <node id="3">
            <word>awarded</word>
            <lemma>award</lemma>
            <pos-tag>V</pos-tag>
            <relation parent="E0">i</relation>
        </node>
        <node id="4">
            <word>the</word>
            <lemma>the</lemma>
            <pos-tag>De</pos-tag>
            <relation parent="6">de</relation>
        </node>
        <node id="5">
            <word>Pulitzer</word>
            <lemma>Pulitzer</lemma>
            <pos-tag>U</pos-tag>
            <relation parent="6">lex-mod</relation>
        </node>
        <node id="6">
            <word>Prize</word>
            <lemma>Pulitzer Prize</lemma>
            <pos-tag>N</pos-tag>
            <relation parent="3">obj2</relation>
        </node>
        <node id="7">
            <word>.</word>
            <lemma.</lemma>
            <pos-tag>U</pos-tag>
            <relation parent="E1">punc</relation>
        </node>
        ...
```

Figure 1: Sample of preprocessed data

```
        ...
        <node id="E0">
            <lemma>fin</lemma>
            <pos-tag>C</pos-tag>
        </node>
        <node id="E1">
            <pos-tag>U</pos-tag>
        </node>
        <node id="E2">
            <lemma>Wilson</lemma>
            <pos-tag>N</pos-tag>
            <relation parent="3">obj1</relation>
            <antecedent>1</antecedent>
        </node>
    </sentence>
</hypothesis>
```

Figure 2: Sample of preprocessed data (continued)

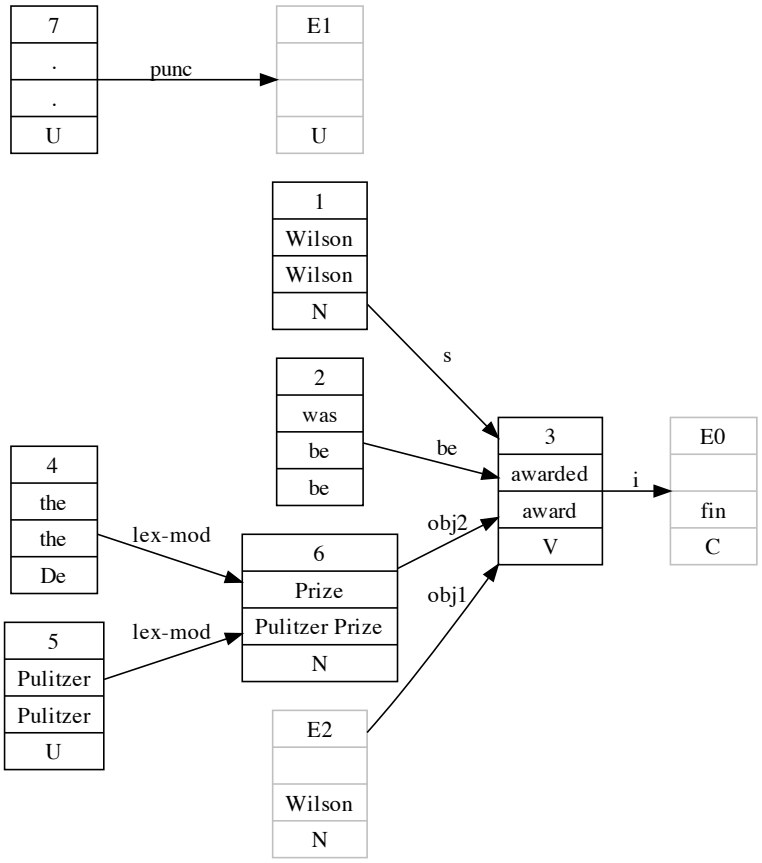

Figure 3: Dependency structure for sample of preprocessed data

Part I: Lexical matching (25 points)

I-a: Word matching (5 points)

Your first task is to implement a simple baseline system that works by matching the words in the hypothesis against the words in the text. The entailment decision relies on how many words from the hypothesis also occur in the text. That is, if all words in the hypothesis also occur in the text, the matching score is one; if none of its words co-occur, the matching score is zero. The entailment decision is based on a threshold value on the matching score. Program an iteration over different threshold values to find an optimal setting.

You receive credit for a word matching system that takes as input the RTE development data and an optimal threshold value, and produces entailment judgements for each T-H pair in the RTE output format as well as the accuracy score.

I-b: Matching lemma and POS tag (5 points)

This improves on the previous system by matching on lemma and/or part-of-speech (POS) tag rather than plain words. You will need to extract the required lemma and POS tag information from the *preprocessed* data. Again iterate over different threshold values to find an optimal setting.

You receive credit for implementing a lemma matching as well as a lemma + POS tag matching system. Both systems must take as input the preprocessed RTE development data and an optimal threshold value, and produce entailment judgements for each T-H pair in the RTE output format as well as the accuracy score.

I-c: BLUE algorithm (10 points)

Implement the BLEU algorithm for multi-level n-gram matching. Apply the BLEU algorithm to calculate a BLEU score for the match between the hypothesis and the text. The entailment decision is based on a threshold value on the BLEU score. Again iterate over different threshold values to find an optimal setting.

You receive credit for a system that takes as input the RTE development data and an optimal threshold value, calculates the BLEU score between hypothesis and text, uses a threshold on the BLEU score to decide on entailment, and produces entailment judgements for each T-H pair in the RTE output format as well as the accuracy score.

I-d: Weighting (5 points)

This is an optional subtask that will get you another 5 points. First write code to calculate the Inverse Document Frequency (IDF) for every term in the development set. Extend the word matching strategy from the first system by weighting each word according to its IDF value. Once again iterate over different threshold values to find an optimal setting.

You receive credit for a IDF-weighted word matching system that takes as input the RTE development data and an optimal threshold value, and produces entailment judgements for each T-H pair in the RTE output format as well as the accuracy score.

Part II: Syntactic matching (25 points)

II-a: Tree edit distance (15 points)

Your main task in this part is to implement the Zhang & Shasha tree edit distance algorithm. This is to be used to calculate the edit distance between the syntactic trees of the text and the hypothesis. Use uniform costs for insertion, deletion and substitution.

You receive credit for a system that takes as input the preprocessed RTE development data, and for each T-H pair prints the correct tree edit distance.

II-b: Entailment judgement (5 points)

Calculate an entailment score based on the tree edit distance. Use a cost of 1 for insertion and substitution and a cost of zero for deletion. Compute the normalized entailment score to compensate for differences in sentence length. Determine the optimal threshold on the normalized score for the development data.

You receive credit for a system that takes as input the RTE development data and an optimal threshold value, calculates the tree edit distance between hypothesis and text, computes the normalized entailment score, and produces entailment judgements for each T-H pair in the RTE output format as well as the accuracy score.

II-c: IDF insertion cost (5 points)

This is an optional subtask that will get you another 5 points. Calculate the Inverse Document Frequency (IDF) for every term in the development set. Now take the IDF value of a word as its insertion cost.

You receive credit for a system similar to the previous one with the exception that it uses the correct IDF value of a word as its insertion cost.

Part III: Machine learning (20 points)

III-a: Feature extraction (5 points)

Extract features from the text and/or hypothesis that allow a machine learning algorithm to learn the entailment relation. The type of features depends on the learning algorithm(s) you intend to use in the subsequent classification step (some algorithms can handle only boolean or numeric features, whereas others can deal with nominal/categorical or even structural features as well).

You receive credit for an implementation that takes as input the RTE development data and outputs a file with extracted feature values for each T-H pair in a format appropriate to a machine learning algorithm implementation of your choice. Your features should minimally represent matching words, matching lemmas, matching lemmas plus POS tag combinations, and matching bigrams.

III-b: Classification (10 points)

Train a machine learning algorithm to recognize textual entailment. You are allowed to use an off-the-shelve implementation of a machine learning algorithm, presumably a classifier, using its command line interface to spawn it as a separate process (alternatively you may link it directly to your own code or use it as server through a socket interface). Examples of machine learning algorithms include Decision Trees, Maximum Entropy Modelling, Instance-based learning, Support Vector Machines, Genetic Algorithms, etc. After training, the induced model should be applied to the development data in order to predict an entailment relation.

You receive credit for an implementation that takes as input the RTE development data, extracts features from it, uses the resulting instances to train a machine learner, applies the learned model to the development data to predict entailment, and produces entailment judgements for each T-H pair in the RTE output format as well as the accuracy score. All of this in a single run without manual steps involved.

III-c: Improved evaluation (5 points)

This is an optional subtask that will get you another 5 points. Implement a better evaluation method, such as cross-validation, leave-one-out or sampling methods, in order to get a more reliable estimate of the accuracy.

You receive credit for an extension of the system developed in the previous step that repeatedly performs training and testing on separate segments of the development data set.

Part IV: Your very own RTE system (30 points)

IV-a: Best system (25 points)

This step gives you a free hand to improve your RTE systems in any way you see fit. Within reasonable limits, of course. Automatically forwarding T-H pairs to human decision maker is obviously not allowed. Likewise, wrapping an existing RTE system built by someone else is not allowed either. If you suspect that a certain extension may be frowned upon, consult the instructors before you start to implement it.

You may continue with the machine learning approach, but if you choose to extend and optimize the lexical or syntactic matching system, or any combination of systems, that is perfectly fine as well. Some possible extensions and optimizations include:

- perform text normalization on dates, times, etc.
- perform Named Entity Recognition (NER) to identify names of persons, companies, countries, etc.
- use English WordNet, eXtended WordNet, UKB or similar resources to lookup synonyms, hyponyms and hypernyms
- use co-occurence features obtained from large text collections[1]
- use Word Sense Disambiguation software to distinguish different meanings of the same word, or use Co-reference Resolution software to resolve the antecedent of anaphoric words (e.g. *he*, *it*, *our*).
- use more training data, for example from RTE challenges other than RTE-2 (but please stay away from *test* data)
- optimize your machine learning approach by applying feature selection, feature weighting and/or optimization of parameters of the learning algorithm (beyond applying build-in optimization facilities, that is, a substantial part of the functionality should be provided by your own code)
- apply fancy machine learning techniques such as bagging, boosting or stacking (again, a substantial part of the functionality should be provided by your own code)
- perform an error analysis: look at the system output, identify common patterns of error, and implement some post-processing to correct them (however, aim for generalization and avoid overfitting)

[1] Querying the web may sound like a good idea for getting co-occurence counts. However, be warned that this may be very slow. Moreover, most search engine will block your access as soon as you start hitting it with automatic queries

You can find more background information on most of these extensions in the lecture notes.

To introduce a little challenge in the spirit of the RTE shared tasks, the number of points awarded for the extended/optimized machine learning approach is calculated according to the following formula:

$$points = \frac{s_{system} - s_{baseline}}{s_{best} - s_{baseline}} \times 25$$

The $s_{baseline}$ score is the baseline accuracy obtained by random guessing (i.e. 0.5 if the test data set is equally balanced). The s_{best} is the highest accuracy obtained by any of the student system(s), which is anywhere between the baseline score and 1.0. The accuracy score of your optimized system, s_{system}, is normalized relative to the baseline and best scores. Points are awarded proportional to this normalized score, with a maximum of 25 points. That is, if your system is (among) the best, you get 25 points; if your system is no better than guessing, you get zero.

A few remarks are in order here. Please don't be evil: optimizing your system on any of the RTE *test* data is considered foul play and will be punished by rewarding you zero points. However, feel free to use any of the RTE *training* data available to improve your system. Furthermore, the instructors reserve the right to apply some common sense under exceptional circumstances.[2]

You should run the system that you developed in Part IV of the project on both the development and test data. The output files for the test data should be named
`<dev|test>_<surname-1>_<surname-2>.txt` where

- `dev|test` is the data set, either `dev` or `test`;
- `surname-1` is the surname of the first student in lowercased ASCII chars; in the case of a team, `surname-2` is the surname of the second student in lowercased ASCII chars.

For example, a valid filename for the output on the test set by students Downing and Marsi would be `test_downing_marsi.txt`.

Result files should be zipped and submitted via email to the designated email address before the deadline, following details announced at the start of the lecture.

[2]For example, if it turns out that all students submit an optimized system with virtually the same score just above the baseline, we will get suspicious and use some other rewarding scheme. Also, if clearly a huge programming effort was made with hardly any result in terms of score, some compensation may still be granted.

IV-b: Report (5 points)

A short report with a maximum of 5 pages of text (plus optional figures and references) must also accompany the project. The report should give a *high level* description of your system *developed in Part IV*, with special focus on:

- the differences between your system and those developed in Part I, II and III
- the motivation for these differences (why you think they are an improvement)
- the most important data structures and the overall control flow used in your code
- a comparison of accuracy scores (in the form of a table) on the development data for your system versus the scores of the systems you implemented in Part I, II and III
- a brief analysis of some common errors made by your system (what went wrong with this pair?)
- any ideas you have on further improving your system (please include motivation)

The report should be written in English. It should not include code, as your code will need to be submitted separately.

Deliverables

Your systems will be demonstrated after working on the project for approximately four weeks (official dates are available on the webpage). For each part of the project for which you intend to get credit, you must show a functioning program which reads RTE data, recognizes entailment, and produces entailment judgements in RTE output data format as well as the accuracy score. All of this in in a *single* run, i.e. no manual steps involved. During the demonstration, the instructor will both (a) look at various parts of the source code, and (b) expect to see each phase running (without repeated crashing). Keeping a list of commands you have to run to demonstrate systems for each part will prevent unnecessary delays. In addition, Part IV requires you to process blind test data and submit the produced output before the deadline. You also need to deliver all your source code and a short report.

If you are working with a partner, **each** person should be capable of answering questions about any part of the code. These will normally be relatively general questions, so its expected that both group members understand the basic functionality of all key components of the system. Failure to satisfactorily answer these questions can result in a loss of points for the project. Group members will receive the same grade for the project.

The combination of demonstration, report and score on the blind test set will be worth a maximum of 100 points. A summary of all subparts described earlier and points for their completion is given in Table 3. Please keep in mind that your time is limited and that it does not pay off to spend all your time on getting every detail of a certain part right and to have no time left for remaining parts. Full credit will be awarded for every reasonably well done part, abstracting from the nasty details that inevitably show up when working with real language data.

Table 3: Grading per part

Part	**Points**
I-a: Word matching	5
I-b: Matching lemma and POS tag	5
I-c: BLUE algorithm	10
I-d: Weighting	5
II-a: Tree edit distance	15
II-b: Entailment judgement	5
II-c: IDF insertion cost	5
III-a: Feature extraction	5
III-b: Classification	10
III-c: Improved evaluation	5
IV-a: Best system	25
IV-b: Report	5