IT3105 – Project 3

Natural Language Processing

Alexandre Balon-Perin

Norwegian University of Science and Technology - NTNU

Academic Year 2011-2012

Table of Contents

Code Structure

The code is divided in different packages. The most important are: Parsing, TextualEntailments, LexicalMaching, SyntacticMatching, MachineLearning and Optimizer.

Parsing

As the name suggest, the Parsing package contains all classes in charge of parsing the xml files. The parsing depends on the input (data preprocessed or not) and on the output wanted (a tree of nodes, a list of words, etc.).

Optimizer

The Optimizer package could contain a collection of different optimization procedures. Only a simple procedure was implemented in order to obtain a good but not perfect approximation of the threshold. The optimizer is initialized with a range of threshold going from 0 to 1 with a step of 0.01. The two thresholds with the best accuracy are saved and used as the minimum and maximum of the range for the next iteration. If there is only one maximum value for the accuracy, the minimum and maximum become values around that particular threshold. The next iteration can start with a new step equals to step/10. The problem is that the algorithm can get stuck in local optimum. For example, if the global optimum is in 0.09 but on the first iteration, the best accuracy were in 0.1 and 0.3, the global optimum will never be found.

TextualEntailments

This package contains the different variants of the objects TextualEntailment. The variants depend on the need to format the raw data extracted by the parser to satisfy the algorithm used.

LexicalMatching

The LexicalMatching package contains all the classes related to the first part of the project. The LexicalMatching class is the parent type of "matcher". Each matcher re-implements the function to satisfy its needs. For example, the BLEU algorithm evaluates the entailment score differently than a word matching.

SyntactidMatching

SyntacticMatching concern the second part of the project. The Tree Edit Distance and the Entailment Judgment are performed here.

MachineLearning

The MachineLearning concern the third and fourth part of the project. It is divided in different sub-packages: FeatureExtraction, Classifier, Learner, AdvancedFeatures. The Classification class is a sort of Main for the Machine Learning calling each step from building the model to classifying the test data. After a quick survey on the Internet to determine which classifier could be good for RTE, KNN, Decision Tree and Naïve Bayes classifier seemed to performed reasonably good. As I did not have the time to try them all, the Naïve Bayes classifier was implemented for its simplicity. The package AdvancedFeatures concerns the fourth part of the project.

Accuracy of the Systems

	Dev Data	Threshold
BLEU	61.25%	0.2879
Tree	52.25%	0.9676
Tree+IDF	53.5%	0.3559
Word	61.75%	0.6239
Word+IDF	58.375%	0.0899
Lemma	60.875%	0.699
Lemma+Pos	50%	0.988
Naïve Bayes	82,375%	/

The word matching seems to perform surprisingly well. A little formatting was applied to avoid words composed of the concatenation of a real word and a punctuation mark. The Lemma accuracy should be higher than the word one but apparently the formatting works quite well. Without formatting, the word matching gets an accuracy of 59%.

Lemma+Pos have surprisingly low accuracy. The reason is that the number of matching is low because the conditions to match are very restricted. As a result, the entailment decision is always NO, which explains the 50% accuracy for a balanced dataset.

The BLEU is also quite good; it was implemented with n-grams for n going from 1 to 4.

The Naïve Bayes classifier seems to be fantastic. However, the development data were used to train so it is not surprising that the accuracy is good. The accuracy of the classifier itself was evaluated around 59% and around 54% with 10-fold cross-validation. This gives a good approximation of the accuracy expected on the test data and this is not very encouraging.

My System

The Naïve Bayes Classifier output the most interesting features in the set. Apparently the Tree Edit Distance was not appropriate. By removing it of the feature sets, the accuracy of the classifier increased from 54 to 55.27%.

Polarity

The polarity feature was added and the resulting accuracy of the classifier increased to 55.41%. As it is a minor change a minor increase was expected. However, it makes sense to add a Polarity feature to the set to counter the following kind of examples:

- Text: "Oliver was walking on the street."
- Hypothesis: "Oliver was not walking on the street."

In this case, the matching is almost perfect and the classifier would classify the example as positive. However, the sense of the hypothesis and sense of the text are opposed. Therefore, the entailment decision should be negative. The polarity is calculated according to the number of matching between the text or the hypothesis and the following list:

Negatives = {"not", "refuse", "wrong", "deny", "no", "false", "ignore", "cannot", "never", "unsuccessfully"}

Unfortunately, this is probably not enough to grasp the complexity of the human language, as we will see in the next part.

Trials

I installed WordNet on my computer and could run it successfully but I had lots of bugs with the interface provided in the Nltk framework and could not use WordNet in my code. I could have parsed the output from the command line interface of WordNet myself but had no time to do so and that felt like reinventing the wheel.

I also tried to add the Decision Tree Classifier from Nltk with the idea of using a boosting technique. I wanted to combine my Naïve Bayes classifier and the Decision Tree but had an unfixable bug in the Decision Tree code provided by Nltk.

In the end, I wasted a lot of time trying to fix bugs in the Nltk and PyML frameworks and problems of compatibilities between the different versions of python. It would have been preferable to implement the machine learning algorithms from scratch.

Finally, using more training data was not an option either as the training data provided on the RTE challenge website are not preprocessed.

Further Improvements

All the trials discussed could have improved the system significantly if they had worked.

Using WordNet would have been really great to compare the words in the hypothesis and text with their respective synonyms and hyponyms. When writing, humans tends to use many synonyms to keep the reader entertain. This is a big problem for our system, which will see two completely different words although they are synonyms and should be considered equals.

The idea of the boosting technique is to use more than one "weak" classifier and give weights to the data instances. A classifier gives more weight to the misclassified instances so that the future classifiers will focus more on those instances.

More training data would definitely increase the accuracy of the classifier because, when classifying a new instance of the test set, the classifier would be more likely to have seen a similar instance in the training set.

Text Normalization is a hard job but it could prevent wrong matching for the following example:

- `Text:` `The order is worth EUR 58 million and handover of the submarines to the Portuguese navy is scheduled for 2010. The contract also includes an option for the same equipment to be built into a third submarine.`
- `Hypothesis:` `The Portuguese Navy has 58 submarines.`

Almost all words match but the idea is again very different. If the currency was normalized to, for example, EUR58.000.000, the matching would not occur but the entailment decision would still most likely be True unfortunately.

Finally, a good idea would have been to try different algorithms like KNN, Decision Tree, etc. to see which one performs the best as a basic RTE classifier.