IT3105 - AI programming

Project III - Textual Entailment

Students

André Christoffer Andersen
Paul Ingebrigt Huse

Date

2011-11-23

Introduction

In this project we have implemented a program to recognize textual entailment, trained on the RTE-2 dataset. Unless otherwise specified, all of our tests are run using 10x cross-validation on this dataset.

In our implementation, we have not separated the different parts of the code formally in to different programs, but instead included the possibility that each part of the code can be run separately from a menu.

General structure of the system

Our system uses instances of the Pair class to contain information. These are constructed out of a text and a hypothesis, which in turn are created out of instances of POS, which contain our words with part of speech data. These are used in the various extractor methods, which include howToMatch as a variable, which tells our extractors how to match the various variables. Our dataextractor method cleans out null values from the dataset (if we tell it to). After this is done, we can run our various extractors on the data. We can also run all of our extractors on the data and dump the results to a file, giving us our final feature set for classifying the final test data. When running our classifiers on the final test data, we use this feature set, read it, and pass it the the constructors for our classifiers. The various methods are seperated into packages depending on their structure: Everything in feature is used to handle the various features of text, everything in extractors is used to extractors is used to find various features, and the rest of our functionality is run from the main rte package.

Part I - Lexical matching

We have implemented the word matching and BLEU algorithm in the FeatureMatching and BLEU classes, in the feature package. Both of these can take either pure text or POS tagged text as input. Feature matching using only words, we identified 470 entailments correctly, giving us an accuracy of 58,75%. Meanwhile, using lemmas, we identified 490 cases correctly, increasing our accuracy to 61,25%. As we can see, using lemmas increased our accuracy by a fair bit. Using BLEU with different numbers of nGrams, we got various results, from 54,5% (with 4-grams) to 56,87% (with 2-grams) accuracy using words. Using lemmas, this was 54,7%.

Weighing: For our IDF algorithm, we have used the tfc*nfx formula, one of the many variants of IDF, as it came recommended in various sources, amongst them “Term Weighing Approaches In Automatic Text Retrieval”, Salton and Buckley, 1987. This gave us a slight increase in accuracy over standard word matching. We compared this to the “classic” IDF algorithm, and the results were as follows: Using weighted feature matching using the tfc*nfx formula, we got 58,75% and 61,13% with words and words + POS tags respectively. With the “classic” IDF algorithm, this was 59% and 60,87%. Since we will primarily be using lemmas and POS tags, we chose the tfc*nfx formula for term weight.

Part II - Syntactic matching

Our implementation of the Zhang & Shasha algorithm includes weighted insertion cost, though it can be run without it, in which case the cost defaults to 1. For our tree structure, we have created an additional “root” node to bind the subtrees together. To avoid it being used in the various operations, we have given operations involving it an arbitrarily high cost.

Part III - Machine learning

Our machine learning methods use the weka library, which is included directly in our system. From it, we used a number of different classifiers. We tested them against each other to try to find the most efficient classifier for our use. For our testing, we used cross-validation. In our quest for the most efficient classifier, we duelled pretty much every possibly classifier in the weka library against each other through a very manual grid search algorithm. By “a very manual grid search algorithm” we mean that we tried out pretty much all of them manually in our code, tweaking their variables and the various features in our dataset until we found which ones seemed to work the best . In the end we ended up combining a neural network, the JRip algorithm and a DecisionTable {NOKO MEIR?} into a voting system, allowing them to vote on the best classification of any given instance. As mentioned above, all of these were evaluated on cross-validated data.

Part IV - Your very own RTE system

Here, we implemented a number of refinements on our original system. We created a

polarization detection method that tries to see if the hypothesis and the text have the same number of negating words. In addition, we also implemented the possibility to use synonyms through WordNet in our word matching algorithm. This is implemented through JAWS (Java API for Wordnet Searching).These are used to find partial matches, or similar words. We hoped that this would increase our accuracy, but it did not do so, at least not by any measure we used. We also included a text normalizer, but as this actually degraded our test results, we decided not to include it when running the test set. In addition, we made a weighted BLEU algorithm. This used the “classic” IDF weighing and increased our accuracy by a small amount.

Error analysis

When looking at a number of errors, they are mostly due to our word matching working badly. For example:
<t>Arsenal sneaked a 1-0 victory over Birmingham at Highbury, taking advantage of Stephen Clemence's own-goal in the 81st minute.</t>
<h>Arsenal lost to Birmingham.</h>

This sentence is generally not correctly analyzed, as too few words match, both in our word matching and bleu analysis. As our tree analysis does not work optimally, this gives our system some problems.

Meanwhile, this sentence is analyzed incorrectly often, due to the opposite:

<pair id="12" entailment="NO" task="IE">
<t>He met U.S. President, George W. Bush, in Washington and British Prime Minister, Tony Blair, in London.</t>
<h>Washington is part of London.</h>
</pair>

Potential for improvement

The results we got when running from our tree analysis were not very good, and we highly suspect that this could lead to great improvement in our results if we managed to use these better. Apart from that, there are a number of additional refinements that we could have done to our system, like co-reference analysis, a working normalizer and similar, but we did not have the necessary time to implement it.

Conclusion

While our system will not be winning any RTE contests any time soon, it does implement several of the methods that “proper” RTE systems use, though it does not have any particular features that might make it competitive against stronger systems.