Machine Learning Experiments for Textual Entailment

Diana Inkpen, Darren Kipp, and Vivi Nastase
School of Information Technology and Engineering
University of Ottawa
Ottawa, ON, K1N 6N5, Canada
{diana,dkipp,vnastase}@site.uottawa.ca

Abstract

We present a system that uses machine learning algorithms to combine features that capture various shallow heuristics for the task of recognizing textual entailment. The features quantify several types of matches and mismatches between the test and hypothesis sentences. Matching features represent lexical matching (including synonyms and related words), part-of-speech matching and matching of grammatical dependency relations. Mismatch features include negation and numeric mismatches.

1 Approach

Recognizing textual entailment between two sentences is a semantic task: one must understand the information conveyed through a sentence, to decide whether the hypothesis sentence contains information we should know based on the information in the test sentence. Nonetheless, heuristics at lexical and lexical-syntactic level were shown to help to some degree (Bar-Haim *et al.*). This paper presents a Machine Learning (ML) system that combined various such shallow heuristics, for the task of recognizing textual entailment (RTE).

We extract match and mismatch features from the two sentences (text and hypothesis). Match features would indicate information overlap, and they include lexical, part-of speech and grammatical relation matching measures. Mismatch features indicate contradictions. We focus on number and negation mismatch scores. We produce feature vectors for all the available development data (RTE2 and RTE1). Weka (Witten and Frank, 2000) is used to train classifiers on these feature vectors. We experiment with the following four machine learning algorithms:

- DT: Decision Trees (J48 in Weka);
- NB: Naïve Bayes;
- SVM: Support Vector Machines (SMO);
- kNN: k Nearest Neighbors (IBK with k=9).

The Decision Trees are interesting because we can see what features were selected for the top levels of the trees. The k Nearest Neighbors algorithm might help when the test pair has similar characteristics to one of the training examples. SVM and NB were selected because they are known to achieve high performance. We experiment with various settings for the ML algorithms including increasing the confidence factor in DT for more pruning of the trees, different kernels for SVM, and different number of neighbors in kNN. After experimenting with various values on cross-validation experiments on the training data, we chose k=9 because it performed best in most settings. We also experimented with combining these classifiers through a meta-learner (voting). We use the prediction accuracies calculated by Weka as confidence scores in the predicted entailments.

Section 2 presents the dependency relation generation output. In section 3 we describe the implemented features. In section 4 we show the results for various classifiers, trained on various training sets, and with various sets of features. Section 5

wraps up the paper with discussion of results and conclusions.

2 Dependency Relations

Establishing connections between the words in a sentence is an important step towards reaching the sentence's meaning. Several systems in previous entailment competitions retrieved these connections as a first step towards verifying the entailment relation.

We use MiniPar, a dependency parser (Lin, 98), to obtain word pairs involved in a grammatical relation. Creating a dependency pair representation for the sentences in the RTE training and test sets is done in two steps:

1. parse the sentences with MiniPar;
2. post-process MiniPar's output.

After these two processing steps, we obtain the following representation:

Paris is the capital of France.

pred be/vbe capital/n
subj capital/n paris/n
of/prep capital/n france/n

The general structure of a generated grammatical dependency tuple is:

Relation Head/POS$_{head}$ Modifier/POS$_{modifier}$

We follow the example above, from the MiniPar output to this final format, to explain and justify the post-processing step. The parse generated by MiniPar (using the lemmatizing and relation-type output options) is presented below:

Paris is the capital of France.

fin C:i:VBE be
be VBE:s:N Paris
be VBE:pred:N capital
capital N:subj:N Paris
capital N:det:Det the
capital N:mod:Prep of
of Prep:pcomp-n:N France

It is interesting to note that when the main verb of the sentence is *be*, MiniPar will consider the predicate to consist of *be* and the verb complement, and it will connect the subject with the complement, bypassing the verb. This is a good feature, as it generates the same dependency pair when a modifier appears as the modifier of the noun, or as complement of the verb *be*. For example, the expressions *interesting paper* and *the paper is interesting* will result in the same dependency pair *paper N:mod:Adj interesting*.

The above parse also shows why we need a post-processing step. First, we filter out pairs such as: *fin C:i:VBE be*, since they are not informative as far as dependency pairs are concerned. We also filter out determiner-noun pairs. Second, we compress two or more dependency pairs, to obtain only pairs containing open-class words (nouns, verbs, adjectives and adverbs). For example, we combine:

capital N:mod:Prep of
of Prep:pcomp-n:N France

to produce the dependency *(of, capital, France)*. This type of compression is performed for dependency pairs containing prepositions, auxiliaries, and clause subordinators and coordinators. For the entailment task, particularly important are negations, as they are one of the easiest to recognize features that can break an entailment relation. A negation is usually connected to the main verb through an auxiliary. MiniPar would produce the following dependency pairs for a negated verb:

Prime Minister John Howard says he will not be swayed by a videotaped warning…
…
sway V:aux:Aux will
will Aux:neg:A not
sway V:be:be be
…

In the post-processing step the auxiliary is bypassed to recover a direct connection between the verb and the negation.

aux sway/v will/aux
neg sway/v not/a

The dependency pairs generated are used:

- to verify dependency pair overlap;
- to treat negation;
- to deal with numbers – in order to detect which entity they modify.

3 Features

We considered 26 features for this task. In many cases we used both the normalized and absolute forms of the same statistics. The normalized scores are between 0 and 1. For example, if the two sentences have 5 content words in common and the hypothesis sentence has 8 content words, the normalized score will be 0.625 (5/8). Normalization is always done with respect to the hypothesis sentence.

Lexical features: Ten of the features involve exact lexical overlap. The Lingua stemmer (Franz & Richardson, 1999) was used to create stemmed versions of words, which were used to compute the following features:

- Stopwords in common (in absolute and normalized form);
- Content words in common (in absolute and normalized form).
- All words in common (in absolute and normalized form)

The text and hypothesis data was tagged using the Lingua part of speech tagger (Coburn, 2005). The tagged data were used to create word match features restricted to nouns and to verbs:

- Nouns in common (normalized);
- Verbs in common (normalized).

We also counted the overlap of tagged words.

Related words: Six features are created using three statistics (synonyms match, verb cause, verb entailment), in both normalized and non-normalized forms, similar to the previous set of features.

When content words do not match, we attempt matching using synonyms in WordNet (Miller, 1995) for all possible senses of the words (since word sense information is not available). This synonym match statistic is considered in both normalized and non-normalized form. It is computed as the sum of the number of words from the hypothesis that directly match words from the text and the number of words from the hypothesis that match a synonym of a word from the text.

Because the task is textual entailment, we compute two verb match statistics using WordNet's *cause to* and *entailment* relations. For each verb pair that groups a verb from the text (v_T) and one from the hypothesis (v_H), we verify if they are in a *cause to* or *entailment* relation, as captured in WordNet:

- verb entailment: v_H *entailment* v_T;
- verb cause: v_T *cause to* v_H.

To generate the features we count the number of verb pairs built in the above forms. A separate count is generated for each. These two counts are taken as features in both normalized and absolute form, thus creating 4 features.

Relations: We compute six features that capture word relations within a sentence. Two such features are created using skip bigrams:

- a normalized count of skip bigrams matches using all words;
- a similar normalized count of skip bigrams created using only nouns and verbs as identified by the Lingua part of speech tagger.

The skip bigram pairs are created by generating a list of all ordered pairs for each sentence (any number of content words can be skipped). These ordered pairs can then be matched between the text and hypothesis sentences. This captures, in a shallow way, the relationships between words in the sentences.

The other four features are generated using grammatical dependency pairs obtained with MiniPar. The dependency relation is expressed through the tuple:

$$Relation \quad Head/POS_{head} \quad Modifier/POS_{modifier}$$

In computing relation overlap, we use the tuple as it is, and also as a dependency pair (*Head, Modifier*), to cover the cases when the same word lemmas appear in different grammatical relations, possibly also with different parts of speech. Tuples and dependency pairs are used to compute the following four features:

- absolute number of overlapping pairs between the test and the hypothesis;
- normalized number of pair overlap (absolute number divided by the number of tuples generated for the hypothesis);
- absolute number of overlapping tuples;
- normalized number of overlapping tuples.

Overall, for the RTE2 development set, there are 769 overlapping dependency pairs, and 709 overlapping tuples; for the RTE2 test set there are 827 overlapping pairs and 761 overlapping tuples.

Mismatch features: These four features were created from information extracted using dependency relation:

- the count of numbers present in hypothesis but not present in the text. This count is normalized by dividing by how many numbers are present in the hypothesis. We attempt to match numbers that appear as modifiers to the same head words, as identified by the dependency relation generation process;
- a normalized count of negated verbs that appear only in the hypothesis and not in the text. Negated verbs in the hypothesis are identified using the dependency relation tuple;
- the number of antonym pairs in the test-hypothesis pair of sentences. For each pairs of words under the same open-class part of speech – one word from the test sentence (w_T), and one from the hypothesis (w_H) – we verify that (w_T,w_H) appears in an antonym relation.
- the normalized version of the previous feature, by the number of content words in the hypothesis sentence.

4 Results

In section 4.1 we present the results of the two runs submitted to the RTE competition.

In sections 4.2 and 4.3 we present experiments that measure the impact of using various classifiers, various subsets of features from the ones computed, and various combinations of the available data sets for training the ML tools used.

4.1 Results for the submitted runs

The University of Ottawa team submitted the results of two runs for the RTE competition.

The first run results were obtained with a classifier trained on the RTE2 development set, with all available features, and tested on the RTE2 test set. The ML algorithm was SVM with non-polynomial kernel, combined with Naïve Bayes. The accuracy obtained by 10-fold cross-validation of this training set was **62%**. The results on task-specific subsets of the RTE2 test set – IE (information extraction), IR (information retrieval), QA (question answering), and SUM (summarization) – are presented in Table 1.

For the IR and SUM tasks the results are between 11% and 27% above the baseline, while for the IE task the system's performance is very close to the baseline. Most likely the examples from the IE subset require deeper semantic features. In the remainder of the paper we report only overall accuracy, not split by tasks. The baseline for all experiments, not included in the tables, is approximately 50%.

For the second run we used more training data: the RTE2 development set; and the RTE1 development, development2, and test sets. The classifier was Naïve Bayes, using all the features. Cross-validation on this training set achieved an accuracy of 57.86%. The results obtained on the RTE2 test set are presented in Table 1.

Some of the relational dependency features were added only after submission (the overlap between dependency pairs and the tuples overlap, normalized and not normalized). Adding them improved the accuracy of classification to **59.25%** on the RTE2 test set, and to 63.25% when doing cross-validation on the RTE2 development set (as seen later in Tables 2, 3, 4, 5, for SVM).

Run	Measure	ALL	IE	IR	QA	SUM
1	Accuracy	**.5800**	.4950	.6100	.5300	.6850
	Avg.prec.	.5751	.4810	.7011	.5465	.7701
2	Accuracy	**.5825**	.5150	.6350	.5050	.6750
	Avg.prec.	.5816	.4904	.7043	.5857	.7514

Table 1. Accuracy and average precision for the two submitted runs on RTE2 test set, including split by task.

4.2 Results for various training sets

We experiment with three different training sets:

1. The RTE2 development set (800 examples).
2. RTE2 development set, plus the RTE1 development, development2, and test sets (2167 examples).
3. RTE2 development; and RTE1 development, development2, and test sets, but keeping only the examples that came from tasks that are included in RTE2 (IE, IR, QA, SUM). The tasks that we filtered out from the RTE1 data were RC (reading comprehension), MT (machine translation), PP (paraphrases), CD (comparable documents). After filtering, the training set contained 1370 examples.

Table 2 presents results for training on each of these three sets, and testing on the RTE2 test set, with four Weka classifiers. In Table 3 the settings are the same as in Table 2, but we report 10-fold cross validation results on the respective training sets. These experiments make use of the full set of features (including the added relational features).

Figure 1 shows how the accuracy increases when we add more training examples. The results are on the RTE2 test set, and the training was done on 200, 400, 600, and 800 examples from the RTE2 dev set.

4.3 Results for various features

Tables 4 and 5 presents results obtained when different subsets of features are used for representing the RTE data. Four classifiers are trained on the RTE2 development set. Table 4 contains results on the RTE2 test set, and Table 5 contains results on the development set by 10-fold cross-validation.

We study the influence of the following types of features:

- Lexical features – exact matching: all words, content words, stopwords, nouns, verbs, etc.;

Training set	DT	NB	SVM	kNN
1. RTE2 dev	.5662	**.5600**	**.5925**	**.5537**
2. + RTE1 all	.5475	.5612	.5825	.5312
3. - selected tasks	**.5700**	.5050	.5800	.5375

Table 2. Accuracy on RTE2 test set for four classifiers trained on different training sets.

Training set	DT	NB	SVM	kNN
1. RTE2 dev	**.5512**	**.5762**	**.6325**	**.5962**
2. + RTE1 all	.5449	.5246	.5736	.5413
3. - selected tasks	.5481	.5138	.5824	.5481

Table 3. Accuracy of 10-fold cross-validation experiments for four classifiers on different training sets.

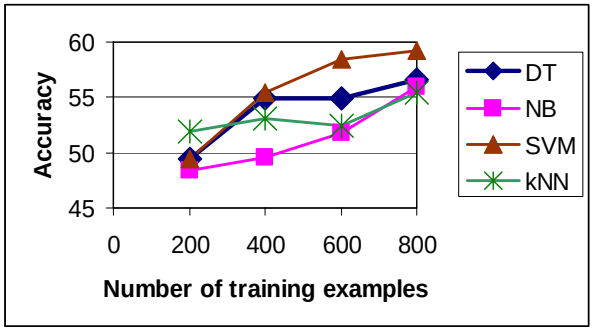

Figure 1. Variation of accuracy with the size of the training set (tested on RTE2 test set, trained on RTE2 dev).

Features	DT	NB	SVM	kNN
Lexical	.5675	.5637	.5775	.5537
Relations	.5575	.5025	.5650	.5487
Mismatches	.5150	.5000	.5150	.5150
Lexical+relations	.5600	.5437	**.5650**	.5487
Lexical+related	.5675	.5637	.5712	.5537
Lexical+mismatches	.5625	.5387	.5812	.5387
Lex.+relations+mism.	**.5687**	**.5687**	.5925	.5450
Lex.+related+mism.	.5662	.5637	.5787	**.5562**
All	.5662	.5600	**.5925**	.5537

Table 4. Effect of various features; training on RTE2 dev and testing on the RTE2 test set.

Features	DT	NB	SVM	kNN
Lexical	.5650	**.6087**	.6187	.6037
Relations	**.5987**	.5025	.5887	.5775
Mismatches	.5125	.5000	.4975	.4900
Lexical+relations	.5620	.5650	.6212	.5887
Lexical+related	.5650	**.6087**	.6187	.6037
Lexical+mismatches	.5662	.6025	.6137	.5700
Lex.+relations+mism.	.5687	.5375	.6312	.6112
Lex.+related+mism.	.5687	**.6087**	.6187	**.6137**
All	.5512	.5762	**.6325**	.5962

Table 5. Effect of various features; all classifiers; training on RTE2 dev and testing by cross validation.

- Related words – approximate matching: synonyms, verb entailment and cause relations;
- Relations – bigrams, verb-noun pairs, dependency relations;
- Mismatches – negation, numbers, antonyms.

5 Discussion of the Results

The impact of the classifier: According to accuracy results, SVM performed best, followed by NB, DT and kNN in no particular order. As seen in Tables 2, 3, 4, 5, there are differences in accuracy between the classifiers, depending on the training set and on the type of features used. Varying the parameters of each classifier or combining classifiers produces only slight improvements.

The impact of the training data: Training on the RTE2 development set seems to be sufficient. Adding all the data from RTE1 (development sets 1 and 2 as well as the test set) helps only if we use examples that come from the same tasks as the ones in the RTE2 data. This is evidenced by the lines 2 and 3 in Table 2, that show a slight improvement in the performance of DT (2.25%), and kNN (0.63%) when the classifiers are trained on the filtered RTE1 data rather than on all RTE1 data, and tested on the RTE2 test set. Curiously, the accuracy of NB goes down.

Although adding data from RTE1 didn't improve the accuracy, the quantity of training data has an influence on the learnability. Figure 1 shows that the accuracy increases for all four classifiers when successively training on 200, 400, 600, and 800 examples from the RTE2 dev set, and testing on the RTE2 test set.

The impact of the features: Exact overlap features achieve slightly lower accuracies than using all the features (Tables 4 and 5). Adding relations to the lexical features seems to help a small amount. Only relational features by themselves achieve good results (except for NB), but lower than lexical match features. Mismatch features added to other features do not seem to have an impact. Probably the mismatches happen in few examples, which were correctly classified using other features. Only mismatch features by themselves achieve baseline performance. Adding related words (approximate lexical matching) does not seem to improve the classification.

6 Conclusion and Future Work

We have tested the use of ML algorithms with data represented using shallow and easy to compute heuristics for the task of recognizing textual entailment. This experimental setup allowed us to test the impact of these heuristics, in various combinations, and with different ML methods. To increase the accuracy of classification, we plan to extract more semantic features, and to improve the quality of some of the current features. In particular the syntactic relations, the negation mismatch, and the number mismatch need more careful examination.

References

Aaron Coburn. 2005. Part-of-speech tagger for English natural language processing, Lingua-EN-Tagger-0.13, CPAN module.

Roy Bar-Haim, Idan Szpektor, Oren Glickman. Definition and Analysis of Intermediate Entailment Levels, *ACL-05 Workshop on Empirical Modeling of Semantic Equivalence and Entailment*. 2005.

Benjamin Franz, and Jim Richardson. 1999. Lingua-Stem-0.81, CPAN Module.

Dekang Lin. 1998. Dependency-based evaluation of MINIPAR. In Proceedings of the Workshop on Evaluation of Parsing Systems at LREC 1998, Granada, Spain.

George Miller, 1995, WordNet: A Lexical Database for English, *Communications of the ACM*, 38(11):39-41.

Ian H. Witten and Eibe Frank. 2000. *Data Mining: Practical machine learning tools with Java implementations*, Morgan Kaufmann, San Francisco, USA.