An Open-Source Package for Recognizing Textual Entailment

Milen Kouylekov and Matteo Negri
FBK - Fondazione Bruno Kessler
Via Sommarive 18, 38100 Povo (TN), Italy
[kouylekov,negri]@fbk.eu

Abstract

This paper presents a general-purpose open source package for recognizing Textual Entailment. The system implements a collection of algorithms, providing a configurable framework to quickly set up a working environment to experiment with the RTE task. Fast prototyping of new solutions is also allowed by the possibility to extend its modular architecture. We present the tool as a useful resource to approach the Textual Entailment problem, as an instrument for didactic purposes, and as an opportunity to create a collaborative environment to promote research in the field.

1 Introduction

Textual Entailment (TE) has been proposed as a unifying generic framework for modeling language variability and semantic inference in different Natural Language Processing (NLP) tasks. The Recognizing Textual Entailment (RTE) task (Dagan and Glickman, 2007) consists in deciding, given two text fragments (respectively called *Text - T*, and *Hypothesis - H*), whether the meaning of *H* can be inferred from the meaning of *T*, as in:

T: "Yahoo acquired Overture"
H: "Yahoo owns Overture"

The RTE problem is relevant for many different areas of text processing research, since it represents the core of the semantic-oriented inferences involved in a variety of practical NLP applications including Question Answering, Information Retrieval, Information Extraction, Document Summarization, and Machine Translation. However, in spite of the great potential of integrating RTE into complex NLP architectures, little has been done to actually move from the controlled scenario proposed by the RTE evaluation campaigns[1] to more practical applications. On one side, current RTE technology might not be mature enough to provide reliable components for such integration. Due to the intrinsic complexity of the problem, in fact, state of the art results still show large room for improvement. On the other side, the lack of available tools makes experimentation with the task, and the fast prototyping of new solutions, particularly difficult. To the best of our knowledge, the broad literature describing RTE systems is not accompanied with a corresponding effort on making these systems open-source, or at least freely available. We believe that RTE research would significantly benefit from such availability, since it would allow to quickly set up a working environment for experiments, encourage participation of newcomers, and eventually promote state of the art advances.

The main contribution of this paper is to present the latest release of EDITS (Edit Distance Textual Entailment Suite), a freely available, open source software package for recognizing Textual Entailment. The system has been designed following three basic requirements:

Modularity. System architecture is such that the overall processing task is broken up into major modules. Modules can be composed through a configuration file, and extended as plug-ins according to individual requirements. System's workflow, the behavior of the basic components, and their IO formats are described in a comprehensive documentation available upon download.

Flexibility. The system is general-purpose, and suited for any TE corpus provided in a simple XML format. In addition, both language dependent and language independent configurations are allowed by algorithms that manipulate different representations of the input data.

[1] TAC RTE Challenge: http://www.nist.gov/tac
EVALITA TE task: http://evalita.itc.it

Proceedings of the ACL 2010 System Demonstrations, pages 42–47,
Uppsala, Sweden, 13 July 2010.

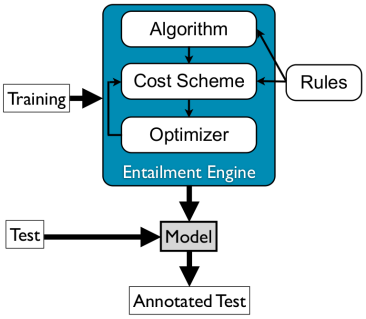

Figure 1: Entailment Engine, main components and workflow

Adaptability. Modules can be tuned over training data to optimize performance along several dimensions (*e.g.* overall Accuracy, Precision/Recall trade-off on YES and NO entailment judgements). In addition, an optimization component based on genetic algorithms is available to automatically set parameters starting from a basic configuration.

EDITS is open source, and available under GNU Lesser General Public Licence (LGPL). The tool is implemented in Java, it runs on Unix-based Operating Systems, and has been tested on MAC OSX, Linux, and Sun Solaris. The latest release of the package can be downloaded from `http://edits.fbk.eu`.

2 System Overview

The EDITS package allows to:

- Create an *Entailment Engine* (Figure 1) by defining its basic components (*i.e.* algorithms, cost schemes, rules, and optimizers);
- Train such *Entailment Engine* over an annotated RTE corpus (containing T-H pairs annotated in terms of entailment) to learn a *Model*;
- Use the *Entailment Engine* and the *Model* to assign an entailment judgement and a confidence score to each pair of an un-annotated test corpus.

EDITS implements a distance-based framework which assumes that the probability of an entailment relation between a given T-H pair is inversely proportional to the *distance* between T and H (*i.e.* the higher the distance, the lower is the probability of entailment). Within this framework the system implements and harmonizes different approaches to distance computation, providing both *edit distance* algorithms, and *similarity* algorithms (see Section 3.1). Each algorithm returns a normalized distance score (a number between 0 and 1). At a training stage, distance scores calculated over annotated T-H pairs are used to estimate a threshold that best separates positive from negative examples. The threshold, which is stored in a *Model*, is used at a test stage to assign an entailment judgement and a confidence score to each test pair.

In the creation of a distance Entailment Engine, algorithms are combined with *cost schemes* (see Section 3.2) that can be *optimized* to determine their behaviour (see Section 3.3), and optional external knowledge represented as *rules* (see Section 3.4). Besides the definition of a single Entailment Engine, a unique feature of EDITS is that it allows for the combination of multiple Entailment Engines in different ways (see Section 4.4).

Pre-defined basic components are already provided with EDITS, allowing to create a variety of entailment engines. Fast prototyping of new solutions is also allowed by the possibility to extend the modular architecture of the system with new algorithms, cost schemes, rules, or plug-ins to new language processing components.

3 Basic Components

This section overviews the main components of a distance Entailment Engine, namely: *i)* algorithms, *iii)* cost schemes, *iii)* the cost optimizer, and *iv)* entailment/contradiction rules.

3.1 Algorithms

Algorithms are used to compute a distance score between T-H pairs.

EDITS provides a set of predefined algorithms, including edit distance algorithms, and similarity algorithms adapted to the proposed distance framework. The choice of the available algorithms is motivated by their large use documented in RTE literature[2].

Edit distance algorithms cast the RTE task as the problem of mapping the whole content of H into the content of T. Mappings are performed as sequences of editing operations (*i.e.* insertion, deletion, substitution of text portions) needed to transform T into H, where each edit operation has a cost associated with it. The distance algorithms available in the current release of the system are:

[2]Detailed descriptions of all the systems participating in the TAC RTE Challenge are available at `http://www.nist.gov/tac/publications`

- Token Edit Distance: a token-based version of the Levenshtein distance algorithm, with edit operations defined over sequences of tokens of T and H;

- Tree Edit Distance: an implementation of the algorithm described in (Zhang and Shasha, 1990), with edit operations defined over single nodes of a syntactic representation of T and H.

Similarity algorithms are adapted to the EDITS distance framework by transforming measures of the lexical/semantic similarity between T and H into distance measures. These algorithms are also adapted to use the three edit operations to support overlap calculation, and define term weights. For instance, substitutable terms in T and H can be treated as equal, and non-overlapping terms can be weighted proportionally to their insertion/deletion costs. Five similarity algorithms are available, namely:

- Word Overlap: computes an overall (distance) score as the proportion of common words in T and H;

- Jaro-Winkler distance: a similarity algorithm between strings, adapted to similarity on words;

- Cosine Similarity: a common vector-based similarity measure;

- Longest Common Subsequence: searches the longest possible sequence of words appearing both in T and H in the same order, normalizing its length by the length of H;

- Jaccard Coefficient: confronts the intersection of words in T and H to their union.

3.2 Cost Schemes

Cost schemes are used to define the cost of each edit operation.

Cost schemes are defined as XML files that explicitly associate a cost (a positive real number) to each edit operation applied to elements of T and H. Elements, referred to as A and B, can be of different types, depending on the algorithm used. For instance, Tree Edit Distance will manipulate *nodes* in a dependency tree representation, whereas Token Edit Distance and similarity algorithms will manipulate *words*. Figure 2 shows an example of

```
<scheme>
  <insertion><cost>10</cost></insertion>
  <deletion><cost>10</cost></deletion>
  <substitution>
    <condition>(equals A B)</condition>
    <cost>0</cost>
  </substitution>
  <substitution>
    <condition>(not (equals A B))</condition>
    <cost>20</cost>
  </substitution>
</scheme>
```

Figure 2: Example of XML Cost Scheme

cost scheme, where edit operation costs are defined as follows:
Insertion(B)=10 - inserting an element B from H to T, no matter what B is, always costs 10;
Deletion(A)=10 - deleting an element A from T, no matter what A is, always costs 10;
substitution(A,B)=0 if A=B - substituting A with B costs 0 if A and B are equal;
substitution(A,B)=20 if A≠B - substituting A with B costs 20 if A and B are different.

In the distance-based framework adopted by EDITS, the interaction between algorithms and cost schemes plays a central role. Given a T-H pair, in fact, the distance score returned by an algorithm directly depends on the cost of the operations applied to transform T into H (edit distance algorithms), or on the cost of mapping words in H with words in T (similarity algorithms). Such interaction determines the overall behaviour of an Entailment Engine, since distance scores returned by the same algorithm with different cost schemes can be considerably different. This allows users to define (and optimize, as explained in Section 3.3) the cost schemes that best suit the RTE data they want to model[3].

EDITS provides two predefined cost schemes:

- Simple Cost Scheme - the one shown in Figure 2, setting fixed costs for each edit operation.

- IDF Cost Scheme - insertion and deletion costs for a word *w* are set to the inverse document frequency of *w* (IDF(*w*)). The substitution cost is set to 0 if a word *w1* from T and a word *w2* from H are the same, and IDF(*w1*)+IDF(*w2*) otherwise.

[3]For instance, when dealing with T-H pairs composed by texts that are much longer than the hypotheses (as in the RTE5 Campaign), setting low deletion costs avoids penalization to short Hs fully contained in the Ts.

In the creation of new cost schemes, users can express edit operation costs, and conditions over the A and B elements, using a meta-language based on a lisp-like syntax (*e.g.* (+ (IDF A) (IDF B)), (not (equals A B))). The system also provides functions to access data stored in hash files. For example, the IDF Cost Scheme accesses the IDF values of the most frequent 100K English words (calculated on the Brown Corpus) stored in a file distributed with the system. Users can create new hash files to collect statistics about words in other languages, or other information to be used inside the cost scheme.

3.3 Cost Optimizer

A cost optimizer is used to adapt cost schemes (either those provided with the system, or new ones defined by the user) to specific datasets.

The optimizer is based on cost adaptation through genetic algorithms, as proposed in (Mehdad, 2009). To this aim, cost schemes can be parametrized by externalizing as parameters the edit operations costs. The optimizer iterates over training data using different values of these parameters until on optimal set is found (*i.e.* the one that best performs on the training set).

3.4 Rules

Rules are used to provide the Entailment Engine with knowledge (*e.g.* lexical, syntactic, semantic) about the probability of entailment or contradiction between elements of T and H. Rules are invoked by cost schemes to influence the cost of substitutions between elements of T and H. Typically, the cost of the substitution between two elements A and B is inversely proportional to the probability that A entails B.

Rules are stored in XML files called Rule Repositories, with the format shown in Figure 3. Each rule consists of three parts: *i)* a left-hand side, *ii)* a right-hand side, *iii)* a probability that the left-hand side entails (or contradicts) the right-hand side.

EDITS provides three predefined sets of lexical entailment rules acquired from lexical resources widely used in RTE: WordNet[4], Lin's word similarity dictionaries[5], and VerbOcean[6].

[4]http://wordnet.princeton.edu

[5]http://webdocs.cs.ualberta.ca/ lindek/downloads.htm

[6]http://demo.patrickpantel.com/Content/verbocean

```
<rule entailment="ENTAILMENT">
  <t>acquire</t>
  <h>own</h>
  <probability>0.95</probability>
</rule>
<rule entailment="CONTRADICTION">
  <t>beautiful</t>
  <h>ugly</h>
  <probability>0.88</probability>
</rule>
```

Figure 3: Example of XML Rule Repository

4 Using the System

This section provides basic information about the use of EDITS, which can be run with commands in a Unix Shell. A complete guide to all the parameters of the main script is available as HTML documentation downloadable with the package.

4.1 Input

The input of the system is an entailment corpus represented in the EDITS Text Annotation Format (ETAF), a simple XML internal annotation format. ETAF is used to represent both the input T-H pairs, and the entailment and contradiction rules. ETAF allows to represent texts at two different levels: *i)* as sequences of tokens with their associated morpho-syntactic properties, or *ii)* as syntactic trees with structural relations among nodes.

Plug-ins for several widely used annotation tools (including TreeTagger, Stanford Parser, and OpenNLP) can be downloaded from the system's website. Users can also extend EDITS by implementing plug-ins to convert the output of other annotation tools in ETAF.

Publicly available RTE corpora (RTE 1-3, and EVALITA 2009), annotated in ETAF at both the annotation levels, are delivered together with the system to be used as first experimental datasets.

4.2 Configuration

The creation of an Entailment Engine is done by defining its basic components (algorithms, cost schemes, optimizer, and rules) through an XML configuration file. The configuration file is divided in modules, each having a set of options. The following XML fragment represents a simple example of configuration file:

```
<module alias="distance">
  <module alias="tree"/>
  <module alias="xml">
    <option name="scheme-file"
```

```
            value="IDF_Scheme.xml"/>
  </module>
  <module alias="pso"/>
</module>
```

This configuration defines a distance Entailment Engine that combines Tree Edit Distance as a core distance algorithm, and the predefined IDF Cost Scheme that will be optimized on training data with the Particle Swarm Optimization algorithm ("*pso*") as in (Mehdad, 2009). Adding external knowledge to an entailment engine can be done by extending the configuration file with a reference to a rules file (*e.g.* "*rules.xml*") as follows:

```
<module alias="rules">
  <option name="rules-file"
          value="rules.xml"/>
</module>
```

4.3 Training and Test

Given a configuration file and an RTE corpus annotated in ETAF, the user can run the **training** procedure to learn a model. At this stage, EDITS allows to tune performance along several dimensions (*e.g.* overall Accuracy, Precision/Recall trade-off on YES and/or NO entailment judgements). By default the system maximizes the overall accuracy (distinction between YES and NO pairs). The output of the training phase is a *model*: a zip file that contains the learned threshold, the configuration file, the cost scheme, and the entailment/contradiction rules used to calculate the threshold. The explicit availability of all this information in the model allows users to share, replicate and modify experiments[7].

Given a model and an un-annotated RTE corpus as input, the **test** procedure produces a file containing for each pair: *i)* the decision of the system (YES, NO), *ii)* the confidence of the decision, *iii)* the entailment score, *iv)* the sequence of edit operations made to calculate the entailment score.

4.4 Combining Engines

A relevant feature of EDITS is the possibility to combine multiple Entailment Engines into a single one. This can be done by grouping their definitions as sub-modules in the configuration file. EDITS allows users to define customized combination strategies, or to use two predefined combination modalities provided with the package,

[7]Our policy is to publish online the models we use for participation in the RTE Challenges. We encourage other users of EDITS to do the same, thus creating a collaborative environment, allow new users to quickly modify working configurations, and replicate results.

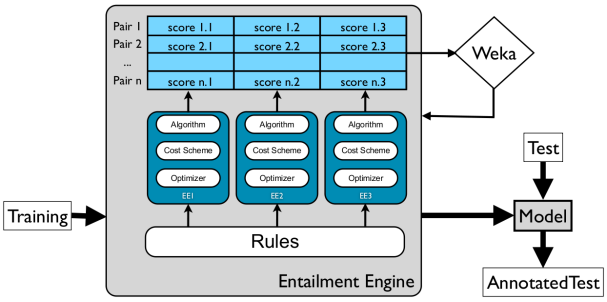

Figure 4: Combined Entailment Engines

namely: *i)* Linear Combination, and *ii)* Classifier Combination. The two modalities combine in different ways the entailment scores produced by multiple independent engines, and return a final decision for each T-H pair.

Linear Combination returns an overall entailment score as the weighted sum of the entailment scores returned by each engine:

$$score_{combination} = \sum_{i=0}^{n} score_i * weight_i \quad (1)$$

In this formula, $weight_i$ is an ad-hoc weight parameter for each entailment engine. Optimal weight parameters can be determined using the same optimization strategy used to optimize the cost schemes, as described in Section 3.3.

Classifier Combination is similar to the approach proposed in (Malakasiotis and Androutsopoulos, 2007), and is based on using the entailment scores returned by each engine as features to train a classifier (see Figure 4). To this aim, EDITS provides a plug-in that uses the Weka[8] machine learning workbench as a core. By default the plug-in uses an SVM classifier, but other Weka algorithms can be specified as options in the configuration file.

The following configuration file describes a combination of two engines (*i.e.* one based on Tree Edit Distance, the other based on Cosine Similarity), used to train a classifier with Weka[9].

```
<module alias="weka">
  <module alias="distance">
    <module alias="tree"/>
    <module alias="xml">
      <option name="scheme-file"
              value="IDF_Scheme.xml"/>
    </module>
  </module>
```

[8]http://www.cs.waikato.ac.nz/ml/weka

[9]A linear combination can be easily obtained by changing the alias of the highest-level module ("weka") into "linear".

```
  <module alias="distance">
    <module alias="cosine"/>
    <module alias="IDF_Scheme.xml"/>
  </module>
</module>
```

5 Experiments with EDITS

To give an idea of the potentialities of the EDITS package in terms of flexibility and adaptability, this section reports some results achieved in RTE-related tasks by previous versions of the tool. The system has been tested in different scenarios, ranging from the evaluation of standalone systems within task-specific RTE Challenges, to their integration in more complex architectures.

As regards the RTE Challenges, in the last years EDITS has been used to participate both in the PASCAL/TAC RTE Campaigns for the English language (Mehdad et al., 2009), and in the EVALITA RTE task for Italian (Cabrio et al., 2009). In the last RTE-5 Campaign the result achieved in the traditional "2-way Main task" (60.17% Accuracy) roughly corresponds to the performance of the average participating systems (60.36%). In the "Search" task (which consists in finding all the sentences that entail a given H in a given set of documents about a topic) the same configuration achieved an F1 of 33.44%, ranking 3rd out of eight participants (average score 29.17% F1). In the EVALITA 2009 RTE task, EDITS ranked first with an overall 71.0% Accuracy. To promote the use of EDITS and ease experimentation, the complete models used to produce each submitted run can be downloaded with the system. An improved model obtained with the current release of EDITS, and trained over RTE-5 data (61.83% Accuracy on the "2-way Main task" test set), is also available upon download.

As regards application-oriented integrations, EDITS has been successfully used as a core component in a Restricted-Domain Question Answering system within the EU-Funded QALL-ME Project[10]. Within this project, an entailment-based approach to Relation Extraction has been defined as the task of checking for the existence of entailment relations between an input question (the *text* in RTE parlance), and a set of textual realizations of domain-specific binary relations (the *hypotheses* in RTE parlance). In recognizing 14 relations relevant in the CINEMA domain present in a collection of spoken English requests, the system achieved an F1 of 72.9%, allowing to return correct answers to 83% of 400 test questions (Negri and Kouylekov, 2009).

[10] http://qallme.fbk.eu

6 Conclusion

We have presented the first open source package for recognizing Textual Entailment. The system offers a modular, flexible, and adaptable working environment to experiment with the task. In addition, the availability of pre-defined system configurations, tested in the past Evaluation Campaigns, represents a first contribution to set up a collaborative environment, and promote advances in RTE research. Current activities are focusing on the development of a Graphical User Interface, to further simplify the use of the system.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under Grant Agreement n. 248531 (CoSyne project).

References

Prodromos Malakasiotis and Ion Androutsopoulos 2007. *Learning Textual Entailment using SVMs and String Similarity Measures*. Proc. of the ACL '07 Workshop on Textual Entailment and Paraphrasing.

Ido Dagan and 0ren Glickman 2004. *Probabilistic Textual Entailment: Generic Applied Modeling of Language Variability*. Proc. of the PASCAL Workshop on Learning Methods for Text Understanding and Mining.

Kaizhong Zhang and Dennis Shasha 1990. *Fast Algorithm for the Unit Cost Editing Distance Between Trees*. Journal of Algorithms. vol.11.

Yashar Mehdad 2009. *Automatic Cost Estimation for Tree Edit Distance Using Particle Swarm Optimization*. Proc. of ACL-IJCNLP 2009.

Matteo Negri and Milen Kouylekov 2009. *Question Answering over Structured Data: an Entailment-Based Approach to Question Analysis*. Proc. of RANLP-2009.

Elena Cabrio, Yashar Mehdad, Matteo Negri, Milen Kouylekov, and Bernardo Magnini 2009. *Recognizing Textual Entailment for Italian EDITS @ EVALITA 2009* Proc. of EVALITA 2009.

Yashar Mehdad, Matteo Negri, Elena Cabrio, Milen Kouylekov, and Bernardo Magnini 2009. *Recognizing Textual Entailment for English EDITS @ TAC 2009* To appear in Proceedings of TAC 2009.