CS-6390: Project Baseline System and Results

Soumya Smruti Mishra (u0926085)
Neetu Pathak (u1006307)

April 19, 2016

Abstract

We are trying to solve one of the interesting problem in NLP called paraphrasing but with twitter data. Given a pair of tweets as input, we would try to infer if they are paraphrases or not. Paraphrase acquisition is a fundamental task in the emerging area of text mining for software engineering. Existing paraphrase extraction methods are not entirely suitable here due to the noisy nature of bug reports. We propose a number of techniques to address the noisy data problem.

1 Task

We present an approach to identifying Twitter paraphrases using simple lexical overlap features. The work is part of ongoing research into the applicability of knowledge-lean techniques to paraphrase identification. We utilize features based on overlap of word, character n-grams and various other distance features and then we train Gradient Boosted Trees(from XGBOOST package).

We used Paraphrase in Twitter (PIT -2015) dataset which was used in SemEval-2015 Task-1. The dataset contains the following files:

1. ./data/train.data (13063 sentence pairs)
2. ./data/dev.data (4727 sentence pairs)
3. ./data/test.data (972 sentences pairs)
4. ./data/test.label (a separate file of labels only, used by evaluation scripts)

Both data files come in the tab-separated format. Each line contains 7 columns:

Topic_Id , Topic_Name , Sent_1 , Sent_2 , Label , Sent_1_tag , Sent_2_tag

1. The "Trending_Topic_Name" are the names of trends provided by Twitter, which are not hashtags.
2. The "Sent_1" and "Sent_2" are the two sentences, which are not necessarily full tweets. Tweets were tokenized by Brendan O'Connor etal.'s toolkit (ICWSM 2010) and split into sentences.

3. The "Sent_1_tag" and "Sent_2_tag" are the two sentences with part-of-speech and named entity tags by Alan Ritter et al.'s toolkit (RANLP 2013, EMNLP 2011).

4. The "Label" column for *dev/train data * is in a format like "(1, 4)", which means among 5 votes from Amazon Mechanical turkers only 1 is positive and 4 are negative.

   We mapped them to binary labels as follows:

   - paraphrases: (3, 2) (4, 1) (5, 0) mapped to 1
   - non-paraphrases: (1, 4) (0, 5) mapped to 0
   - debatable: (2, 3) which we discard while training binary classifier

5. The "Label" column for *test data* is in a format of a single digit between 0 (no relation) and 5 (semantic equivalence), annotated by expert.

   We mapped them to binary labels as follows:

   - paraphrases: 4 or 5 mapped to 1
   - non-paraphrases: 0 or 1 or 2 mapped to 0
   - debatable: 3 which we discarded in Paraphrase Identification evaluation

   We discarded the debatable cases in the evaluation of Paraphrase Identification task.

We output our test labels in the same format is given by the dataset to compare the results easily. We do not output any data files as all out evaluation and training is done inside the same python script.

2 System Architecture

Text pre-processing is essential to many NLP applications. It may involve tokenizing, removal of punctuation, PoS-tagging, and so on. For identifying paraphrases, this may not always be appropriate. Removing punctuation and stop words, as commonly done for many NLP applications, arguably results in the loss of information that may be critical in terms of PI. We therefor keep text pre-processing to a minimum as suggested by Asli Eyecioglu and Bill Keller.

2.1 Feature Extraction/Selection

Most of our code consists of various types of feature extraction. We created features according to two previous works on this data set. (papers can be found in external resources)

As the basis for deriving a number of overlap features, we consider different representations of a text as a set of tokens, where a token may be either a word or character n-gram. For the work described here we restrict attention to word and character unigrams, bigrams and trigrams. In particular, word bigrams may provide potentially useful syntactic information about a text. Character bigrams, on the other hand, allow us to capture similarity between related word forms. Possible overlap features are constructed using basic set operations:

2.1.1 Basic semantic Features:

- **String Features**, that indicate whether the two words, their stemmed forms and their normalized forms are the same, similar or dissimilar i.e, we created precision, recall and F-measure features out of these. We used the Morpha stemmer (Minnen et al., 2001),3 Jaro-Winkler string similarity (Winkler, 1999) and the Twitter normalization lexicon by Han et al. (2012).
- **Size of union:** the size of the union of the stemmed and non-stemmed tokens of words, bi-gram and tri-gram in the two texts of a candidate paraphrase pair.
- **Size of intersection:** the number of stemmed and non-stemmed tokens of words, bi-gram and tri-gram common to the texts of a candidate paraphrase pair.
- **Text Size:** the size of the set of tokens, bi-grams and tri-grams representing a given text.

2.1.2 Counting Features

We generated counting features for tokens, bigrams and trigrams. For some of the counting features, we also computed the ratio of these.

1. *Basic Counting Features:*
   - Count of n-gram.
   - Count & Ratio of Digit: count & ratio of digits in q_i, t_i, and d_i.
   - Count & Ratio of Unique n-gram
2. *Intersect Counting Features*
   - Count & Ratio of $sent\_1$'s n-gram in $sent\_2$'s n-gram
3. *Intersect Position Features*
   - Statistics of Positions of $sent\_1$'s n-gram in $sent\_2$'s n-gram
     For those intersect n-gram, we recorded their positions, and computed the following statistics as features.
     - minimum value (0% quantile)
     - median value (50% quantile)
     - maximum value (100% quantile)
     - mean value
     - standard deviation (std)
   - Statistics of Normalized Positions of $sent\_1$'s n-gram in $sent\_2$'s
     These features are similar with above features, but computed using positions normalized by the length of $sent\_1$.

2.1.3 Distance Features

1. **Jaccard coefficient**

$$\text{JaccardCoef}(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

2. **Sorensen-Dice distance** We used two forms of this feature one from the python library distance and other based on the below formula.

$$\text{DiceDist}(A, B) = \frac{2|A \cap B|}{|A| + |B|} \tag{2}$$

   are used as distance metrics, where A and B denote two sets respectively. For each distance metric, two types of features are computed.

3. **Levenshtein distance** is a string metric for measuring the difference between two sequences. Used the python's library distance's implementation.

4. **Cosine similarity** is a measure of similarity between two sentence of an inner product space.

These distance measures were applied on all the tokens/unigrams, bi-grams and tri-grams.

2.1.4 TF-IDF Based Features

This may be also called bag of words feature extraction. We extracted various TF-IDF features from the original sentence $sent\_1$ and $sent\_2$ and stacked them with other features before classification.

2.1.5 Non-numerical Feature Conversion

XGBoost(our classifier) doesn't (yet) handle categorical features automatically, so we need to change them to columns of integer values. We used LabelEncoder from sklearn library to transform non-numerical labels to numerical labels. There were many non numerical labels like trendid, sent_1, sent_2, sent_1_tags, sent_2_tag and all the n-gram features. So instead of removing them after extracting other features from them we kept it and converted them to numeric values so that our classifier can treat them as features.

3 Classifier, Experimental Design and Evaluation Metrics

Our classifier was a form of Gradient Boosting Trees (GBT). We chose this one, as considerable work had been done using SVM and Logistic Regression in this area. Also we were familiar with Decision Trees and Ensembles from our Machine Learning class and also had implemented variants of them for that course.

GBT is essentially an ensemble technique which uses smaller decision trees. The classification problem for our case was to classify if a set of sentences as paraphrase or not.

We used the python package XGBoost which is short for eXtreme gradient boosting. This is a library that is designed, and optimized for boosted (tree) algorithms. The goal of this library is to push the extreme of the computation limits of machines to provide a scalable, portable and accurate for large scale tree boosting.

We experimented with various parameters of our classifier such as max_depth, learning_rate and n_estimators while training the classifier. This was done using GridSearch.

The best parameters we obtained were max_depth=4, learning_rate=0.01 and n_estimators = 100.

We did a 3-fold cross-validation while training our data and the accuracy we got was 64.6%.

After predicting our test labels we used accuracy, precision, recall and F-measure as metrices to evaluate our results and we got the following results.

4 Evaluation Results and Future work:

1. **Precision:** 0.7982
2. **Recall:** 0.52
3. **F-Measure Score:** 0.6297
4. **Accuracy:** 87.23%

In this work we have mainly concentrated on extensive semantic based feature extraction and we a total of around 300 semantic features. In the next cycle we would like to work upon the effect of comparing triplet (subject predicate and object combination) in the given tweets pair in improving results over basic lexical similarity techniques. Also in this work part-of-speech and named entity tags didn't have much role to play. So we have to consider those when designing our final implementation.

We would also like to do some feature evaluation of our present features and throw away all the irrelevant features. We haven't looked into this in this iteration but hope to do so in the next.

5 Team Members Contribution:

Both the team members had equal contribution in this work as most of the work here is related to feature extraction. While Soumya was assigned with extraction of Count based, TFIDF and non numerical features, Neetu was assigned with extraction of basic semantic features and distance features. Rest of the code for classification and import export of data was very simple and was done by Soumya.

6 External Resources and Links:

We didn't use any external data set but we might also use Microsoft Research Paraphrase Corpus and PPDB (the Paraphrase Database) if required.

Papers Refereed: Most of our feature selection and extraction are inspired from the following papers and write-ups.

1. ASOBEK: Twitter Paraphrase Identification with Simple Overlap Features and SVMs by Asli Eyecioglu and Bill Keller.
2. Extracting Lexically Divergent Paraphrases from Twitter by Wei Xu, Alan Ritter, Chris Callison-Burch, William B. Dolan and Yangfeng Ji
3. Solution for the Search Results Relevance Challenge by Chenglong Chen $(https://github.com/ChenglongChen/Kaggle_CrowdFlower/blob/master/Doc/Kaggle\_CrowdFlow$

Libraries or Packages Used:

1. Classifier: $https://github.com/dmlc/xgboost$
2. Tools: http://scikit-learn.org/stable/
3. and, http://www.nltk.org/