Paraphrase Detection in Tweets

I. Introduction

Paraphrase detection is the process of determining whether two sentences are similar in meaning. It is a vital task that aids several NLP applications such as machine translation, plagiarism detection, text summarization, information extraction, etc. Text summarization in particular is directly aided by paraphrase detection, since it involves condensation of the text and removal of redundant sentences. We approach this problem through comprehensive analysis of the syntactic structure as well as the semantic content of the sentences using deep learning.

Social networks such as Facebook and Twitter heavily use NLP-based techniques for use cases like content suggestion and ”Trending” modules, among others. They mainly deal with noisy, user-generated data. Therefore, work in this field will directly benefit the millions of daily users of these services. Consequently, we attempt the more difficult task of applying paraphrase detection to inherently noisy data - tweets. An example of two sentences which are worded differently but are actually paraphrases would be

’Bailey and Austin playing for the Rams’
’Tavon Austin and Stedman Bailey both to the Rams’

II. Related Work

Madnani et al. utilized a set of machine translation metrics to achieve an f-measure of 84.1% on the MSRP paraphrase corpus[3]. Socher et al. used recursive autoencoders with dynamic pooling to measure word- and phrase-similarity between sentences and obtained an f-measure of 83.6%.[4]. Although the methods we use in our paper are similar to the ones proposed in [4], we attempt to apply this to a relatively new Twitter paraphrase corpus [6], which is a more complex problem.

Xu et al. used a novel latent variable model that jointly models paraphrase relations between word pairs and sentence pairs to achieve an f-measure of 72.4% [6]. Additionally, Das et al. implemented a logistic regression model on the same data set and achieved an f-measure of 60% [2]. We implement a sliding window approach and compare our performances with the above model as baseline. We additionally implement normalization [5] on tweets as a pre-processing step, which has not been explored in the above papers, and show that it improves the performance of the proposed classifier.

III. Approach

We formally define the paraphrase detection problem thus-given two input sentences, determine whether the two sentences are similar in meaning or not.

An outline of our approach is as follows. First, in a pre-processsing step, we employ Tweet normalization methods to reduce the ungrammatical and noisy nature of tweets. This will improve the performance of the steps that follow. Next, we obtain word embeddings for every word in our data corpus. A word embedding is an n-dimensional feature vector (of fixed size for all words) that attempts to incorporate the semantic meaning of the word. Once we have the word embeddings, we attempt to compute phrase representations because semantic similarity between two phrases provides high signal with respect to paraphrase similarity. At this stage, we attempt two approaches - (1) Unfolding recursive autoencoders (RAE) using parse trees by Socher et al. and (2) Sliding window representation of phrases. Once we have word and phrase representations of each sentence, the next step is to compute similarity between them. The problem here is that the two sentences are not of equal length and so we employ dynamic pooling to convert them into a fixed size representation. We also augment this representation with linguistic features such as sentence length, placeholder frequency, etc. Finally, given a fixed-size similarity matrix between two sentences we use this as a feature vector passed to a logistic regression classifier to classify whether it’s a paraphrase or not.

A. Tweet Normalization

Since we work with a Twitter dataset, the data is not always syntactically correct. In order to ameliorate this issue, we employ Tweet Normalization [5]. This involves 2 main corrections. The first is the conversion of informal vernacular slangs to regular dictionary equivalents, for instance,‘bruh’ and ‘dawg’ into ‘brother’ and ‘friend’ respectively. The second is typo-correction, such as the conversion of ‘muuuccchhh’ and ‘grandmotha’ to ‘much’ and ‘grandmother’ respectively.

B. Word Embeddings

Word embedding representations are essentially vectors that capture the semantic and lexical properties for various words. A word embedding described by $\mathbf{W} : \mathbf{words} \rightarrow \mathbb{R}^n$ is a paramaterized function that maps words to high-dimensional vectors of 100 or so dimensions. For example:

$$W(“artificial'') = [0.3, -0.1, ...0.9]$$

These embeddings can be learnt from a neural language model. The dimensions of the vector can be set, and every word in the language can be represented by a embedding vector of that dimension. We use the word2vec package in Python to obtain 100-dimensional word embeddings for our vocabulary of normalized Twitter words. But word embeddings work best when they have been trained over a large, meaningful dataset to learn the common contexts in which words are used. Hence, we use a set of 67-dimensional word embeddings which were trained over the corresponding Wiki pages of those words [1].

C. Phrase Representations

Once we get the word embeddings, the next step is to figure out how to use them to represent phrases. We identified 2 ways to do this - Unfolding Recusive Autoencoders and Sliding Window.

1) Unfolding Recursive Autoencoders (RAE): This is the method that Socher et al. [4] used to obtain their final word and phrase representations. The initial phrase representations are obtained by concatenating the embeddings of constituent leaf nodes in its parse tree. Note that the concatenation is with the constituent leaf nodes in the parse tree, so these nodes may not always appear together in the original sentence as you go higher up in the parse tree. Recursive autoencoders are used to improve these phrase representations.

A recursive autoencoder is a deep learning neural network model that learns the parameters of the model through backpropagation. It uses the syntactic parse tree structure of sentences. The idea is to construct the vector representations of non-terminal nodes in the parse tree (phrases) from the constituent children, and then reconstruct back the children to compute the reconstruction error. To construct the parent node from the constituents we apply,

$$p = f(W_e[c1; c2] + b) \tag{1}$$

where f is an activation function, in this case tanh, W is the N X 2N parameter matrix that we want to learn, [c1;c2] is the concatenation of the word embedding vectors of the constituent words, and p is the final phrase representation.

Once we construct the vector for the phrase, we break the phrase down again to the constituent words by splitting the vector, and then minimise the distance between the representations. The paper uses unfolding recursive autoencoders as an improvement to the recursive autoencoder. In this case, the reconstruction error is computed with not just the children of the tree but with the other nodes in that sub tree. The sum of all the reconstruction errors is computed and minimised using gradient descent to get the final parameter values. The code for doing the above is available online, so we were able to plug in our test set, but due to limitations in the code, we were not able to modify the parameters for training the recursive autoencoder. Moreover, computing reconstruction errors in unfolding recursive autoencoders is an immensely time-consuming process, so we instead implement the sliding window module described in the next section.

2) Sliding Window: The sliding window module is also used to obtain phrase representations from word embeddings. In this case, similar to the last one, we construct word embedding vectors for normalized twitter words.

We apply a sliding window model on our word embedding vectors for each sentence to obtain phrase vectors. We take a sliding window of lengths 1 through 5 to get the phrase vectors of corresponding lengths by concatenating the embedding vectors in that window. Using this instead of parse trees ensures that we get accurate phrase representations of words that appear together.

D. Similarity Matrix

We use the word and phrase vectors as obtained in the previous section to construct a similarity matrix for a pair of sentences. To ensure that similarity matrices for each pair of sentences are of the same shape, we employ dynamic pooling as described in [4].

E. Dynamic Pooling

Our similarity matrix at this stage has dimensions $m \times n$, where m and n are the lengths of the sentences. We need to bring this to a fixed-size $n_p \times n_p$ matrix where n_p is the pooled size. We do this by coarsening the $m \times n$ grid into a $n_p \times n_p$ grid. We divide the original grid into roughly n_p parts and take the minimum of each $\lfloor m/N_p \rfloor \times \lfloor n/n_p \rfloor$ grid and use that value as the representative in the coarsened grid. The reason that we use the minimum value is that we're looking for minimum Euclidean distances as a measure of similarity. Hence, using other functions like averaging may obscure the similarity. This method is similar to the approach followed by [4]

F. Linguistic Feature Additions

We finally compute a number of linguistic features to augment the dynamically pooled similarity matrix from the previous step. Promising results were seen with the addition of the following features for each sentence - sentence length, common named-entity count (number of common named entities present in the sentences) , number of placeholder words (numbers, punctuation marks).

G. Logistic Regression

Here, we flatten the dynamically pooled matrix and add the aforementioned linguistic features to generate our final feature vector for training. We use a logistic regression model to perform the final classification.

IV. Evaluation

Our data set [6] consists of 11,513 pairs of tweets with 35% of the datapoints being paraphrases and the rest marked as non-paraphrases. The data was annotated using Amazon Mechanical Turk and it has good inter-annotator agreement score. We used a 60-40 train-test split, and used 10-fold cross-validation to tune our parameters. We finally report the performance of the best classifier on the test set.

Since the data set is skewed in favor of the negative class, we evaluate the performance of our classifier using f-measure.

Das et al.[2] implemented a logistic regression model on the same Twitter dataset and achieve an f-measure of 60%. We consider this to be our **baseline** model.

V. Results

A. Window size

We employ the sliding window as a method to capture context at a phrase level. One parameter for this is the size of the sliding window. We need to ensure that the window is large enough to capture context, yet small enough to avoid the noise prevalent in large phrases. We perform cross-validation on the Twitter data set with normalization to find the best window size. We used values in the range $1 \rightarrow 5$.

Window size	1	2	3	4	5
f-measure	39.7%	40.0%	36.2%	37.1%	34.8%

From the above table, we find that a spanning window of size 2 (corresponding to bigrams) gives the best f-measure.

B. Pooled size

We use dynamic pooling to reduce the similarity matrix to a fixed dimension n_p. Socher et al used cross validation to find the best n_p and found it to be 15. Our initial hypothesis was that the best value for our corpus would be shorter as tweets are, on an average, shorter than the sentences in the MSRP corpus. To validate this, we perform cross-validation on the original Twitter dataset with normalization to find the best n_p. The range of values we tested on are from $5 \rightarrow 15$. The window size parameter is set to 2.

After cross validation, We find that the best performing pool size n_p for our dataset is 12 with an f-measure of 47.0%.

C. Twitter Normalization & Linguistic Features

Our hypothesis was that normalizing the tweets would improve both the syntactic parse trees and the word embeddings that we trained. In the table below, we show that normalization indeed improves the performance, for the recursive autoencoder and the Sliding Window Model (SW). In addition, we show that as each feature set is added to our model, the f-measure score improves. The best f-measure score is obtained with a combination of the sliding window model and the three feature sets we used: Sentence Length (SL), placeholder word frequency (PW), and number of common named entities (CNE) between the two input sentences.

Comparing Performance of different Models on normalized and non-normalized data

Model	Not Normalized	Normalized
RAE	26.0%	28.3%
Sliding Window (SW)	35.3%	47.0%
SW+PW	37.41%	49.04%
SW+PW+SL	43.33%	54.68%
SW+PW+SL+CNE	**54.73%**	**63.82%**

D. Roundup

We see that normalization does indeed improve the classifier. Using the tuned parameters from above, we compute the f-measure on the held-out test set. Our best model (sliding window with linguistic features) performs well on the test set giving us an f-measure of 63.82 %. We manage to **better the baseline classifier**'s f-measure score of 60 %.

VI. Discussion

As we expected the unfolding RAE with dynamic pooling model used by [4] performs poorly on the twitter dataset. This is because the recursive auto encoders they used were trained on the MSRP data set and hence struggled with noise prevalent in tweets. We hypothesized that using twitter normalization would help remove some of the noise and this was borne out by our results for both the RAE and the sliding window models.

We found that training the word embeddings on a large corpus is quite important. The word embeddings trained using data from Wikipedia out performs the embeddings learned only from the train set. This is because the wikipedia dataset is far more extensive in terms of the words covered and also captures the contexts in which the words are used better than the twitter corpus.

The sliding window model succeeds in capturing some context, and we see that a windows size 2 performs the best in capturing the context and at the same time lowering the influence of noise.

We tried several Linguistic features in addition to the word embedding vectors we obtained from the neural Language model. The feature that helped the most was the number of common Named Entities between the two sentences. This is expected, if two sentences are talking about the same person or location, it is a strong feature that they might be similar.

Future work on this problem would include implementing the RAE model proposed by [4] and training the model on the twitter corpus. We, can also consider add twitter specific features like emoticons and hashtags to improve the performance of the classifier, we could not implement these features in our current model due to limitations of the dataset.

References

[1] Al-Rfou, Rami, Bryan Perozzi, and Steven Skiena. "Polyglot: Distributed Word Representations for Multilingual NLP." *Proceedings of the Seventeenth Conference on Computational Natural Language Learning. Association for Computational Linguistics.*

[2] Das, Dipanjan, and Noah A. Smith. "Paraphrase identification as probabilistic quasi-synchronous recognition." *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1. Association for Computational Linguistics, 2009.*

[3] Madnani, Nitin, Joel Tetreault, and Martin Chodorow. "Re-examining machine translation metrics for paraphrase identification." *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Association for Computational Linguistics, 2012.*

[4] Socher, Richard, et al. "Dynamic pooling and unfolding recursive autoencoders for paraphrase detection." *Advances in Neural Information Processing Systems. 2011.*

[5] Xu, Wei, Alan Ritter, and Ralph Grishman. "Gathering and generating paraphrases from twitter with application to normalization." *Proceedings of the Sixth Workshop on Building and Using Comparable Corpora. 2013.*

[6] Xu, Wei, et al. "Extracting Lexically Divergent Paraphrases from Twitter." *Transactions of the Association for Computational Linguistics 2 2014.*