CS221 Project Report
Classifying Paraphrases in Twitter

Melvin Johnson Premkumar
Dept. of Computer Science
Stanford University
Email: melvinj@stanford.edu

Isaac Caswell
Dept. of Computer Science
Stanford University
Email: icaswell@stanford.edu

Zane Silver
Dept. of Electrical Engineering
Stanford University
Email: zsilver@stanford.edu

I. Abstract

Our project is detecting and identifying related paraphrases in Twitter, as part of the SemEval challenge for 2015. Given two sentences (Tweets), we determine whether they express the same, or very similar, meaning. Then we assign the two phrases a degree score between 0 and 1 that indicates our level of confidence their relatedness. The training dataset is provided by semEval and contains about 17,790 annotated sentence pairs, and comes with tokenization, part-of-speech and named entity tags. The testing dataset consists of a further 1k examples from a different time period that has been annotated by an expert. SemEval also provides several baselines against which to compare our algorithm's performance.

TODO: Report the Results and Error Analysis from using Google Wordvec and Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection for this task; and discuss how to improve upon them. Maybe move the "Results" subsection up here?
We are working with Xiao Cheng, a PhD candidate in the Computer Science Dept.

II. Introduction

A. Task Review

Given two phrases, we have to determine whether they express the same or very similar meaning. The input to the system are two sentences from Tweets in Twitter. The system has to predict whether they express the same meaning and also produce a similarity number between 0 and 1.

B. Oracle

The oracle for our twitter phase classifier are human identifiers. This relies on humans reading two different twitter phrases and determining whether or not they refer to the same context. The added constraints are that the phrase identification must be performed in a timely manner (under a minute per phrase pair) and without outside help (i.e. searching on the internet the meaning of a word, phrase, etc.). Using the members of our group as the oracle, we found that we have a 99.98% accuracy rate. The oracle fails to accurately identify all phrases, or misidentifies some phrases as being related, due in part to human error, and in part to the fact that whether two sentences are paraphrases does not always have a clear answer.

C. Challenges in Paraphrasing & Analysis of Tweets

Accurately identifying related paraphrases is a challenging process and extensive NLP research focuses on this topic. The sources of difficulty arise from the panoply of meanings that can be extracted from words or phrases. The mapping of words and phrases to meanings is not one-to-one but rather one-to-many. The correct meaning of a phrase depends on many dynamic variables such as time, location, length of text, context, and syntax.

Tweets pose a particular challenge. Not only is the meaning of a tweets hyper sensitive to time and location, but tweets also frequently contain linguistic errors and idiosyncratic styles [1]. Additionally, the limit in the number of characters of a tweet constrains the ability to provide contextual meaning to the phrase. For this reason, other indicators become more important in deciphering a tweet's meaning: time, location, slang, etc. These are some of the challenge of classifying tweets.

There exist many techniques to ameliorate some of the NLP difficulties inherent in tweet classification. This includes correcting incorrect spelling before processing and improving the library of English parts-of-speech [1]. Additionally, it helps to understand the use of tweets and the specific characteristics of different uses. For example, using tweets for political news updates is much different than celebrity gossip or rumors [2]. These, along with traditional phrase classification, are used to properly identify related tweets.

III. Data description and Preprocessing

A. Preprocessing step: normalization of the data

Twitter datasets tend to attract slang and misspellings, arbitrary capitalization and the like. For this reason we normalize our data first with TODO. We have found an external library (twitter lexicon normalization) which implements X, Y, by means of Z; we will apply this to our dataset and see what the effect on our performance is. This tool

can be used with a Twitter specifc entity detector, available online at https://github.com/sem-io/python-twitter-spell-checking. Furthermore one could experiment with basic word stemming.

B. Corpi Used

The corpi we used for thie project were the following: (TODO: are these really corpuses? Should I say 'external datasets'?)

- Pretrained word vectors publically from Google's word2vec project. These were trained on the Google News dataset (about 100 billion words) with the continuous Skip-gram model described by Milikov et al (http://arxiv.org/pdf/1310.4546.pdf). The model contains 300-dimensional vectors for 3 million words and phrases, from Agence France Press English Service, Associated Press Worldstream English Service, The New York Times Newswire Service and The Xinhua News Agency English Service.
- The English Gigaword corpus (released January 28, 2003), an archive of about 1200M words from newswire text data in English that has been acquired over several years by the Linguistic Data Consortium.
- Glove vectors from http://nlp.stanford.edu/projects/glove/ (on the twitter dataset)
- wikipedia dump + giga word corpus

IV. Baseline Description

As a baseline model we found the MULTIP (Multi-Instance Learning Paraphrase) model for the Twitter phrase classifier. This model was proposed by Xu et. al [3]. Their model serves as a good starting point and is the reference baseline model for our twitter classifier project. The model description below is a summary of the proposal for the MULTIP model.

The advantage of using the MULTIP model is that it relies on sentence level relations using a feature-based classifier. Extensive research has been undertaken to improve upon the MULTIP model so that it can correctly identify two related twitter texts. This has proven to be especially difficult because of the amount of variability in the type of tweets and the prevelence of generalized naming for entities. Specifically, the paper presenting MULTIP uses the example sentences:

- That boy Brook Lopez with a deep 3
- brook Lopez hit a 3 and I missed it

The above example is just to illustrate difficulty that arises when the named entities are generalized into different words.

Firstly, the MULTIP model relies on the at-least-one-anchor assumption. The at-least-one-anchor assumption is derived from the idea that twitter messages posted around the same time and the same topic share lexical paraphrases. This intuition allows us to extend the idea further: that most related tweets, not just those posted around the same time or location, will share a lexical paraphrase. The lexical paraphrases may be the same words, or different words, that are contained in two different tweets that identity the tweets are being related. In other words, it is assumed that related tweets contain an anchor, a lexical paraphrase. In the context of CS 221, an anchor is simply a factor.

The MULTIP model sets up a learner that observes the labels on groups of sentence-level paraphrases. This is due to the at-least-one-anchor assumption described early. This method contrasts with a learner that observes word pairs. There are now two layers in the model, since the sentence-level analysis inherently relies on the word-level analysis. This two-level hierarchy is described in explicit detail in the proposal by Xu et. al [3]. In the context of this class, the two-level model is simply a factor graph with paraphrases/sentences as the upper nodes and the word-pairs as the lower, or leaf, nodes.

For each pair of sentences there is a binary variable y that represents whether the two sentences are related. This is determined to be true ($y = 1$) if there is at least one anchor found in common between the two sentences. We determine if at least one anchor exists in the set of word-pairs between the two sentences. The set of word-pairs is the set of unique word-pairs that can be formed from one word in each sentence. For each word-pair there is another binary variable z which determines if those words are an anchor or not. Therefore, for $y = 1$ there must be at least one $z = 1$ in the set of word-pairs for the two sentences.

For two sentences s_1 and s_2, y(s_2,s_1) = 1 if z_j = 1 for at least one j. z_j = 1 only if w_j = (w_{j1}, w_{j2}) is an anchor, where w_{ji} corresponds to a word in sentence s_i. For every sentence pair (s_i, s_k), there will be $|s_i|$ * $|s_k|$ word-pairs, where $|s_i|$ is the number of words in sentence s_i.

This model suffices as a basic outline on how to solve the Twitter classifier. However, this is only our baseline as the actual model and algorithm we implement is described below.

V. Approaches

A. Foundations: Dense word vectors

Our approach to solving the problem of Twitter paraphrases hinges on the application of a dense vector representation for words and phrases. This section details the idea behind this approach, and its advantages.

In standard applications of machine learning techniques to NLP, words have been represented by "one-hot" vectors, which are vectors whose entries are all zero except for a single 1 entry. A typical way of representing a document, for instance, is as a counts vector, or with some derivative of word counts, such as tf-idf (term frequency-inverse document frequency). This approach yields a vector of length $|V|$ for each document, where entry i corresponds to some function of the count of the ith word in the vocabulary. Such a count vector can be seen as a sum of vectors for each word, where each vector is a one-hot vector with its one entry in the index corresponding to that token. These models have

been moderately successful, but suffer from some serious problems. Primarily, a one-hot word representation doesn't offer any information about the semantics of an individual word, and ignores the local context in which the word appears. Furthermore, as a result, any reasonable measure of vector similarity between two words, with such a model, is zero.

A one dimensional representation doesn't match what we know about words; words can be similar in certain ways, and different in others. The words "fear" and "trembling", for instance, are similar in that they are both nouns, different in that one is emotional and the other physical; close to each other in negative valence and further from each other in commonness in standard parlance, and so on; a human could easily identify hundreds of dimensions in which they are close or far. This is precisely the idea behind dense vector representations of words. Each word in these models is represented as a dense vector w, where each entry w_k corresponds to some 'topic' or property of that word. (One may think w as an unnormalized probability distribution over different attributes of the word.) In practice, to generate these distributions, some form of statistical clustering is performed first, usually based on which words tend to co-occur. Each dimension ('topic') is determined by the percent membership to different clusters, based on the global word co-occurrence matrix X [7].

There are a variety of advantages fo this approach. First of all, and germane to our application to the Twitter dataset, there is now a well defined and meaninful distance between words. Distance metrics such as the cosine and Euclidean distance between two dense word vectors can give a relatively meaningful approximation of how similar these words are as a whole, which we demonstrate to be of practical use in determining similarity of tweets.

The expressive power of word vectors is especially evident in the example of vector similarity, a metric which respects a finer-grained relationship between words. For example, vector distance between words makes intuitive sense in analogy detection: the analogy king is to queen as man is to woman should be encoded in the vector space by the vector equation vec("king") vec("queen") = vec("man") vec("woman"). Indeed, Milikov et al's model demonstrates that the vector obtained by the equation vec("Madrid") - vec("Spain") + vec("France") is closer to vec("Paris") than to any other word vector, and furthermore that even simple vector addition can produce meaningful results: for example, vec("Russia") + vec("river") is close to vec("Volga River"), and vec("Germany") + vec("capital") is close to vec("Berlin"). That the semantics of these phrases can be brought out by such simple compositionality has exciting implications for the ability to understand language using basic mathematical operations on the word-vector feature space.

B. Method 1: Wordvec and Dense Vector Representation of Words

Our first approach the problem of Twitter paraphrase detection was to train word vectors with Wordvec, Google's implementation of the Skip-gram algorithm developed by Mikilov et al in Google [5], and assign a similarity score between two tweets that is an aggregate of the similarities between individual words in the tweets. More precisely, we define the distance metric D_{tweets} for two tweets t_1 and t_2, as the following:

$$D_{tweets}(t_1, t_2) = \sum_{w_i \in t_1} \sum_{w_j \in t_1} D_{cosine}(vec(w_i), vec(w_j))$$

where

$$D_{cosine}(x, y) = \frac{x^T y}{\|x\| \, \|y\|}$$

and $vec(x)$ is the vector representation of word x, as learned by the Skip Gram model, which one will now detail. TODO: do we learn the cutoff?

The idea behind Skip-gram is to learn word vector representations that are good at predicting nearby words. Skip-gram furthermore models each word as having both an 'input' and an 'output' form, which reflects the directional relation between word pairs. Specifically: given a sequence of training words $w_1, W_2, w_3 .... w_T$, the objective of the Skip-Gram model is to minimize the average log probability:

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} logp(w_{t+j} | w_t)$$

where c is the training context, which gives the size of the neighborhod of of w_t to look at, and which may depend on w_t. The size of c can be tuned to trade off accuracy (a higher c value) with efficiency (lower c). Specifically, Skip Gram defines $p(w_{t+j}|w_t)$ using the softmax function:

$$p(w_O | w_I) = \frac{exp(v'^T_{wo} v_{wI})}{\sum_{w=1}^{W} exp(v_w^T v_{wI})}$$

Where v_w and v'_w are 'input' and 'output' representations of w and W is the number of words in the vocabulary. This expression, however, is in general computationally infeasible; as an alternative, in practice, Mikilov et al use Hierarchical softmax as an approximation of the softmax function, which defines the probability of one node given abnother by a random walk down a binary tree of the output layer with the W words as the leaves, and represents the probability of child nodes explicitly for each word. The math will be excluded here for the sake of brevity. With this framework in place, any standard optimization technique can be used. I this case we look into stochastic gradient descent.

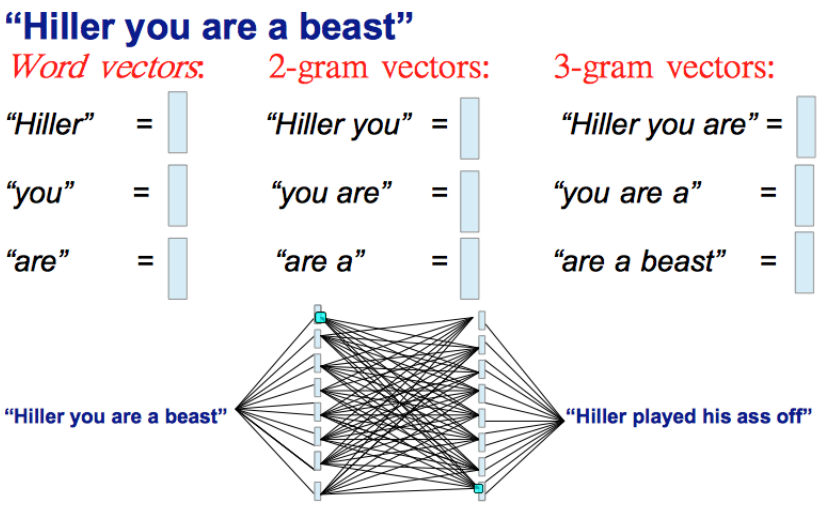

Fig. 1. Word Vectors

Using this model on words alone, we get the undesireably low accuracy of 64%, which does not compare well to the baseline of 72%. We therefore extend this model by learning vectors for bigrams and trigrams as well, and including phrase vector averaging. What this means is that we represent each sentence (tweet) not only as a vector over word vectors, but also over word vector objects that correspond to the bi- and trigrams in the sentence. The modified similarity function then becomes the following:

$$D'_{tweets}(t_1, t_2) = \sum_{n=1}^{n=3} \sum_{\substack{\phi_i \in t_1 \\ |\phi_i|=n}} \sum_{\substack{\phi_j \in t_1 \\ |\phi_j|=n}} D_{cosine}(vec(\phi_i), vec(\phi_j))$$

Where ϕ_i of length n is an n-gram in a tweet.

1) Example: Consider the input D, where D_T is a corpus containing T words, $w_1, W_2, w_3 ....w_T$. For concreteness we can say that D is a corpus of N tweets sampled from Twitter in May 2013. We now break each tweet into a vector of its words and its bigrams and trigrams (which we term 'phrases'), each of which will be assigned a numerical vector representation, which we will now calculate. We begin with a random initialization over the vector for each word/phrase, the dimensionality of which we determine based off of empirical results. Then, until we have reached convergence, we compute the gradient of the log likelihood (defined above), parametrized by Hierarchical softmax, and update our word vectors. When convergence comes, we have a word vector representation for each word and each phrase. Using these dense vector representations, we can now compare tweets. We iterate over all pairs of phrases between the two tweets, and take the cosine difference between those phrases. The similarity score of the two tweets is now the average of these distances of pairs. If the similarity score is high, we say that they are paraphrases; if it is low we say that they are not.

Each tweet can furthermore be assigned a vector representation itself, by aggregating its word vectors in a hierarchical way. There is an intuitive interpretation that we can now glean from these aggregate vectors. By observing which words have high values for different elements in the vector, we can determine the semantic meaning of each dimension in the vector. For instance, one might find that all tweets w_k dealing with sports have a high value for w_{k5}, whereas tweets about social rights have a high w_{k18}. This would indicate that the fifth dimension of the word-vector space corresponded to the general semantic topic of sports, and the 18th to issues of social justice.

C. Method 2: Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection

In this section, we describe a state-of-the-art work [4] on using dynamic pooling and unfolding recursive autoencoders (RAE) for paraphrase detection.

1) Model: The model used here are Neural networks. Neural networks are especially useful in automatically learning features from data. This is much more powerful than manually specifying features to the classifier. A very powerful idea called neural language models was first introduced by Bengio et al. [**?**]. The idea is to jointly learn an embedding of words into an n-dimensional vector space and these vectors can be used to predict how likely a word is given its context. A word embedding matrix $L \in \mathbf{R}^{n \times |V|}$, where $|V|$ is the size of the vocabulary is obtained by running gradient descent on the network. Once this training is done we can obtain a word's vector as just a column in L.
The goal of autoencoders is to learn a representation of their inputs. In this experiment we used recursive autoencoders to

learn representations for sentences. The autoencoder uses a neural network layer to compute the parent representation. We then decode the vectors of the children in a reconstruction layer and compute the Euclidean distance between the two as the reconstruction error. For each of the children, we recursively compute the reconstruction loss for their children as well. The goal of the training is to minimize the reconstruction error across all inputs pairs.

2) Algorithm: For a given sentence, a parse tree is obtained initially. A binary parse tree for a sentence is of the form of triplets of the parents with children: $(p \rightarrow c_1 c_2)$. Each child can either be an input word vector x_i or a nonterminal node in the tree. We now can compute parent representations using its two children (c_1, c_2) using a neural network layer:

$$p = f(W_e[c_1; c_2] + b) \tag{1}$$

where $[c1; c2]$ is the concatenation of the two children, f is an tanh activation function and $W_e \in \mathbf{R}^{n \times 2n}$ the encoding matrix that we want to learn.

To assess these n-dimensional representation of a parent p we decode the vectors of its direct children into a reconstruction layer and compute the Euclidean distance between the original input and its reconstruction:

$$[c_1'; c_2'] = f(W_d p + b_d) \tag{2}$$

$$E_{rec}(p) = ||[c_1; c_2] - [c_1'; c_2']||^2 \tag{3}$$

The training objective is the minimization of all reconstruction error of the input pairs at nonterminal nodes p in a given parse tree T:

$$E_{rec}(T) = \sum_{p \in T} E_{rec}(p) \tag{4}$$

The unfolding recursive autoencoder is the same as the standard RAE with the only difference being that a reconstruction is created for the entire spanned subtree under each node. The reconstruction error is now computed as a concatenation of all the leaf nodes beneath a non-terminal node.

We now use a dynamic pooling pooling method to convert a similarity matrix S generated by sentences of lengths m and n to a matrix S_{pooled} of fixed length $n_p \times n_p$. The idea is to partition the rows and columns in S into n_p roughly equal parts. S_{pooled} is defined as the matrix of the minimum values of each rectangular region within this grid.

Finally, a classifier is trained on this matrix which takes as input a matrix and returns a binary decision of whether the two sentences are paraphrases or not.

3) Example: Given two sentences and their parse trees T_1 and T_2, the unfolding recursive neural network will first compute features for these two sentences using the algorithm described above. We then compute the euclidean distance between all word and phrase vectors of the two sentences. This will form the S matrix. We then compute the S_{pooled} matrix as stated above. When this is given to the classifier, it will make a decision of whether the sentences are paraphrases or not.

D. Method 3: GloVe representations

A recent exciting development in the class of algorithms using dense word vector representations is GloVe, developed at Stanford by Pennington, Socher and Manning. It addresses the problem that standard co-occurrence models are succeptable to noise: The vast majority of co-occurrences in the matrix X will tend to be ones that happen very rarely, and without much particular semantic significance. Even just the 0 entries of X will tend to account for 75-95% of data. Furthermore, some words that co-occur extremely commonly have little semantic relevance, such as "the" along with any noun. To compensate for this, Pennington et al introduce a weighting function into the least-squares error which lowers the impact of common co-occurrences as well as the least common ones. Furthermore, by ensuring that the function approaches 0 as co-occurrence approaches 0, one naturally excludes all pairs which do not co-occur in the corpus from the calculations, which seems intuitively to make sense, and has the benefit of speeding up calculation significantly. Pennington et al demonstrate that, as a result of using only nonzero co-occurrence values, these methods run in approximately $O(|C|^{0.8})$, where $|C|$ is the size of the corpus. The objective they propose furthermore claims to address several problems with loss functions of other unsupervised methods based on co-occurrence, such as skip-gram and ivLBL. [6]

E. Method 3: SVM with features

TODO(Isaac) - write me Building off the previously described methods, we construct a SVM classifier using word vectors as features, in addition to extra features, including:

- Entities discovered with NER (Named Entity Recognition)

We try this with two different kernels, namely linear and RBF (radial basis function). The RBF kernel,

The RBF kernel can also be seen as a low band pass filter, a technique used for instance in image processing to smooth images.

Method	Precision	Recall	F1
test1	0.501	0.547	0.326
test2	0.726	0.751	0.74

VI. Results

	Baseline	Method 1	Method 2 (RAE)
F1	0.501	0.547	0.326
Acc	0.726	0.751	0.74

VII. Error Analysis

TODO(Melvin) - write me

VIII. Conclusion

TODO(melvin) - write me

IX. Future Work

There are several things we can do to improve the accuracy of our current model. Following is a list of the next things we will work on

References

[1] Leon Derczynski, Allan Ritter, Sam Clark, Kalina Bontcheva, Twitter Part-of-Speech Tagging for All: Overcoming Sparse and Noisy Data. RANLP 2013.

[2] Brendan OConnor, Michel Krieger, David Ahn, Tweetmotif: Exploratory Search and Topic Summarization for Twitter. ICWSM 2010.

[3] Wei Xu, Alan Ritter, Chris Callison-Burch, William B. Dolan, Yangfeng Ji, Extracting Lexically Divergent Paraphrase from Twitter. 2014 Association for Computational Linguistics.

[4] Richard Socher, Eric H. Huang, Jeffrey Pennington , Andrew Y. Ng, Christopher D. Manning, Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection, NIPS 2011.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffry Dean Distributed Representations of Words and Phrases and their Compositionality, 2013

[6] Jeffrey Pennington, Richard Socher, Christopher D. Manning GloVe: Global Vectors for Word Representation

[7] Richard Socher and Christopher Manning Deep Learning for NLP NAACL 2013, Atlanta