Paraphrasing 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

***Abstract*—Our project is detecting and identifying 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. 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. SemEval also provides two baselines against which to compare our algorithm's performance. We experiment with different word vector representations trained using different techniques. We also experiment with different classifiers for the task, and perform feature selection to select the best features. Our final best system obtains a precision of 0.665, recall of 0.422, F1 of 0.517 and an accuracy of 0.75. We worked in collaboration with Xiao Cheng, a PhD candidate in the Computer Science Dept.**

I. Introduction

Given two phrases, we determine whether they express the same or very similar meaning. The input to the system is two sentences from tweets in Twitter. The system predicts whether they express the same meaning. System performance is evaluated by SemEval primarily by the F-1 score and Accuracy against human judgements. This is the definition of the SemEval task which we address in this project.

Paraphrase detection as a field has recently a lot of attracted interest, with the hope that solving the task might help shed light on how best to model the semantics of sentences. For computational linguists, solving this problem may shed light on how best to model the semantics of sentences. For natural language engineers, the problem bears on information management systems like abstractive summarizers that must measure semantic overlap between sentences [1], question answering modules [2] and machine translation [3] [13].

As with any task which deals with the semantics of natural language, accurately identifying related paraphrases is very difficult. It is not enough to determine that two sentences refer to the same thing, which is already a nontrivial task; they must also refer to the same thing in the same context and relation to the other variables in the sentence. The meaning of a sentence, especially a short one such as a tweet, is also heavily dependant on the context it was written in. With our dataset, for instance, time, place and knowledge of current events (especially sports) play a large role in understanding of a tweet. Not only is the meaning of a tweet hypersensitive to time and location, but tweets also frequently contain linguistic errors and idiosyncratic styles [4]. 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 may become more important in deciphering a tweet's meaning: time, location, slang, and so on.

There are several ways of ameliorating some of these difficulties, including normalizing the data (e.g. correcting spelling, stemming) and improving the library of English parts-of-speech [4]. 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 [5]. The data provided by SemEval has already been preprocessed to a certain degree, dealing with some of these issues. Nonetheless, the hardest part remains: dealing with the semantics.

It is worthwhile to note that although paraphrase detection is a growing field, thus far no work has been done with paraphrase detection in Twitter specifically. With this in mind, we review the field of distributed representations for words, and detail the algorithms we apply to the problem. These together serve as a literature review.

We first discuss the training and testing data used, including preprocessing of the data. The oracle and baseline will then be detailed. Following this we describe the underpinnings and capabilities of distributed word representations (dense word vectors), and why they make sense for this problem. With this in place, we describe our method for comparing two tweets, and how to learn word vectors with the Skip-gram model. Thereafter we describe three other approaches we use, namely Unfolding Recursive Autoencoders, GloVe vectors, and a classifier with features derived from word vectors and other sources. Finally, we discuss results, give an error analysis, conclude, and suggest directions for future work.

II. Data description and Preprocessing

A. Train and Test Data

Our training data (provided by SemEval) consists of 18k representative tweets semi-randomly collected from Twitter's feed, annotated by several Mechanical Turkers. The data is balanced (appx. 35% paraphrase, 65% non-paraphrase), and have already been tokenized and split into sentences. Part

	sentence pairs	fraction paraphrase	fraction not
train data	13063	.35	0.65
dev data	4727	0.31	0.69

	Precision	Recall	F1	Accuracy
Oracle	0.60	0.91	0.74	0.76
Baseline	0.70	0.39	0.50	0.73

of speech (POS) tagging is also provided. Hashtags were removed. Dev data is also provided, collected from the same time period and the same trending topics. The test set consists of a further 1k data from a different time period, annotated by an expert. Table II-A summarizes this.

B. 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 standard normalization scripts which remove special characters. We have found an external library (twitter lexicon normalization) which groups common misspellings tweets to a single word. In essence, it clusters misspellings into groups. This tool can be used with a Twitter specific entity detector, availiable online at [15].

C. Additional Training Corpora Used

We used three different corpora to train our word vectors:

- **googleNews**: a dataset of articles from Google News, containing about 100 billion tokens and 692K types (after removing words ocurring less than 5 times). [8]
- **wiki_giga.txt**: Aka Gigaword, the English Gigaword Fifth Edition corpus (released June 17, 2011), an archive of 4.3 billion tokens from newswire text data in English that has been acquired over several years by the Linguistic Data Consortium at the University of Pennsylvania. [12]. Contains text from theEnglish service of the Associated Press Worldstream, the Central News Agency of Taiwan, the Los Angeles Times/Washington Post Newswire, the Washington Post/Bloomberg Newswire and the New York Times Newswire and Xinhua News Agency.
- **wiki_giga_skipgram.txt**: the combination of Gigaword5 and Wikipedia2014, a Wikipedia dump. The combined dataset has 6 billion tokens. [8]
- **twitterGlove**: a dataset of about 2 billion tweets, with 27 billion tokens total. [9]

D. 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.). To test the performance of the oracle, we tested one of our own members on 100 tweet pairs. His performance compared to the baseline (discussed in the next section) is summarized by the following table:

The table highlights how difficult the task is, in that even a human evaluator does not significantly outscore the baseline. The tweets are short and without context (e.g. *"angie shud go and i want candice to win"* and *"Now it s up to Candice and Amber"*), and the criteria for paraphrase can be ambiguous. For instance, *"am i the only one not getting amber alerts"* and *"I need a amber alert to wake tf up in da mornings Ha"* are both about Amber Alerts, but are about different aspects of them, so it is unclear if they should count as paraphrases.

III. 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 [6]. 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 prevalence 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 [6]. 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.

IV. Foundations: Distributed word representations

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

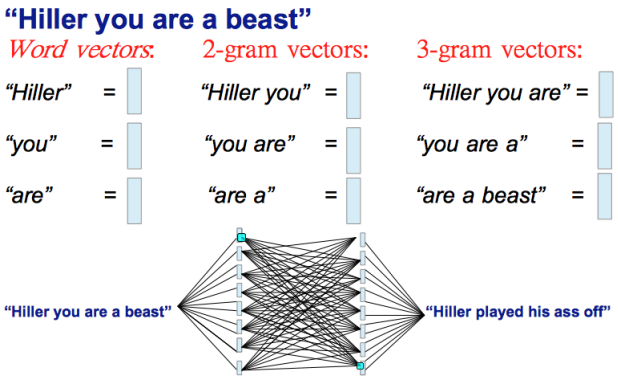

Fig. 1. Word Vectors

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 [10].

There are a variety of advantages of this approach. First of all, and germane to our application to the Twitter dataset, there is now a well defined and meaningful 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 compositionality (i.e. adding and subtracting vectors), 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

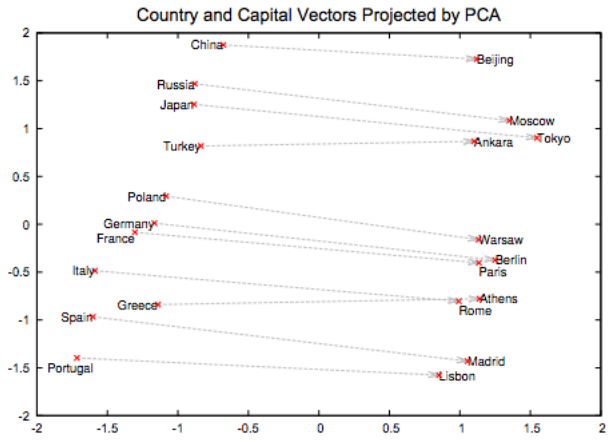

Fig. 2. Compositionality in Word Vectors

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. This idea, illustrated in IV, is important to our classifier.

V. Approaches

A. Word vectors using distributed representations

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 [8], 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:

$$D_{tweets}(t_1, t_2) = \sum_{w_i \in t_1} \sum_{w_j \in t_1} D_{cosine}(vec(w_i), vec(w_j)) \quad (1)$$

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 detail in the next paragraph. A pair of tweets is then predicted to be a paraphrase if this distance is above a certain threshold. We learn this threshold using grid search, which is an exhaustive search over all values applicable to a variable with a constrained range.

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:

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} logp(w_{t+j} | w_t) \quad (2)$$

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:

$$p(w_O | w_I) = \frac{exp(v'^T_{wo} v_{wI})}{\sum_{w=1}^{W} exp(v'^T_w v_{wI})} \quad (3)$$

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.

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:

$$D'_{tweets}(t_1, t_2) = \sum_{n=1}^{n=3} \sum_{\substack{\phi_i \in t_1 \\ |\phi_i| = n}} \sum_{\substack{\phi_j \in t_1 \\ |\phi_j| = n}} D_{cosine}(vec(\phi_i), vec(\phi_j)) \quad (4)$$

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 above our learned threshold, we say that they are paraphrases; otherwise 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.

B. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection

In this section, we describe a state-of-the-art work [7] 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:

$$p = f(W_e[c_1; c_2] + b) \tag{5}$$

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:

$$[c_1'; c_2'] = f(W_d p + b_d) \tag{6}$$

$$E_{rec}(p) = |||[c_1; c_2] - [c_1'; c_2']||^2 \tag{7}$$

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

$$E_{rec}(T) = \sum_{p \in T} E_{rec}(p) \tag{8}$$

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. Figure V-A1 gives a clear example of the process visually.

C. GloVe distributed 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

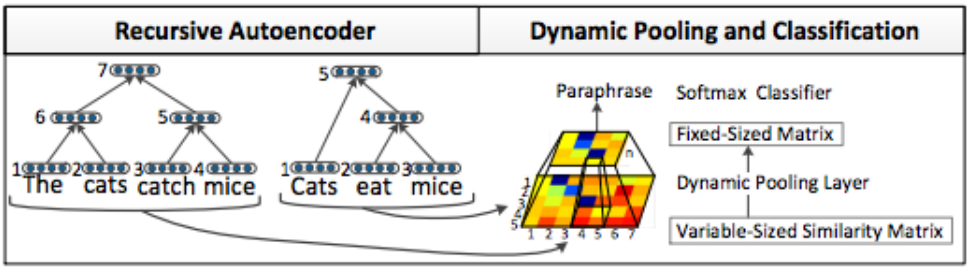

Fig. 3. An overview of RAE. The recursive autoencoder learns phrase features for each node in a parse tree. The distances between all nodes then fill a similarity matrix whose size depends on the length of the sentences. Using a novel dynamic pooling layer we can compare the variable-sized sentences and classify pairs as being paraphrases or not [7].

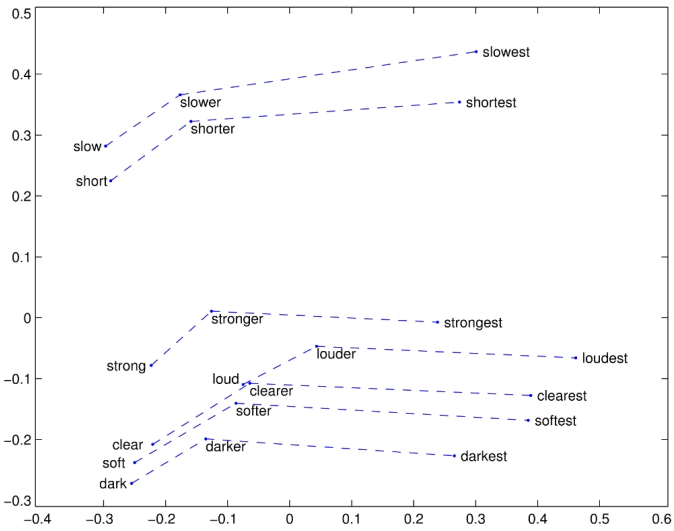

Fig. 4. A PCA projection demonstrating word compositionality using GloVe vectors with comparative and superlative forms: the spacial relationship between the learned vectors reflects the semantic relationship between the words. [9]

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, in [9] the authors 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. [9] 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 [9]. Figure V-B3 shows an example of word relationships learned by vectors trained by GloVe.

D. Classifier with word vector features and other features

Building off the previously described methods, we construct a SVM classifier using features derived from word vectors as features, in addition to other more standard NLP features. The features we used in total are the following, for a pair of tweets (t_1, t_2):

1) L1 similarity between t_1 and t_2 (from word vectors)
2) Cosine similarity between t_1 and t_2 (from word vectors)
3) Count of matching NER tags
4) 1-gram (word) overlap
5) 2-gram overlap
6) 3-gram overlap

In order to calculate the first two features, we use any of the above-described methods to calculate word- and phrase vectors associated with a tweet. We then average these vectors to make a single vector to represent the tweet as a whole. As previously mentioned, this technique makes semantic sense, as addition of word vectors simulates composition of their meaning; this final vector is thus a reasonable representation for the meaning of the tweet as a whole. Once we have a single vector representation for each tweet, we can apply standard distance metrics to the pair, resulting in features 1 and 2 above.

To obtain the third feature, we use the Stanford NER (named entity recognition) tagger to assign NER tags to words in each tweet, and then simply count how many tags are shared between the two. This is a simple and effective way to get an idea of if they are discussing the same topic, which is naturally important in paraphrase detection. Note however that this alone is not sufficient to determine paraphrase: the tweets *I never appreciated the connection between #affinefunctions and #triangleinequality before I knew the grt. and benfcnt. Mr. Goldberg, #224W* and *dayum Adam G so tall and hand-some* may both be correctly identified as referring to the entity "Adam Goldberg", but are about different topics and shouldn't be classified as paraphrases.

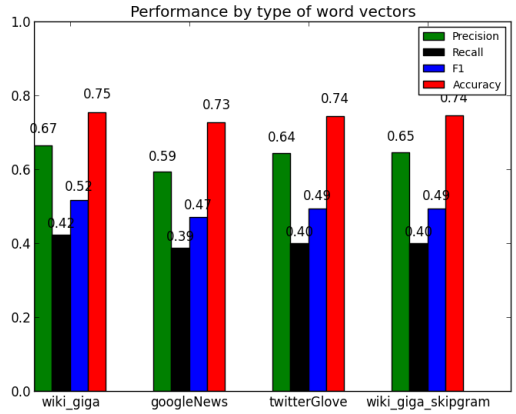

Fig. 5. Performance on Different word vectors

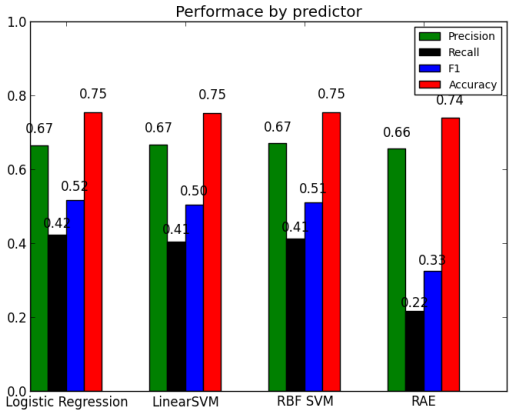

Fig. 6. Performance for different predictors

The final three features are counts of exact n-gram overlap. This is more or less as it sounds. The 2-gram overlap of two tweets, for instance, is a count of how many pairs of consecutive words they share. These features help capture similarity in topic, as with the NER features, and furthermore help capture the interaction between topics, which is what tripped us up in the last example (where the topic "Adam Goldberg" was shared between the two tweets, but was interacting with different topics himself.)

We run this classifier with both the linear and RBF (radial basis function) kernel. The difference between these two is not easy to explain intuitively, and not worth explaining mathematically for the purposes of this paper; but it may help to think of the RBF kernel as a low band pass filter, a technique which is also used for instance in image processing to smooth images [11]. We also use a logistic regression classifier with addition to the SVMs.

VI. Results

In this section, we present the results of our system on various system combinations. Figure 5 shows the results on various distributed representations. As described in the Section II-C we trained distributed representations on four different corpora. It can be seen that all the corpora achieve similar performance but the best performance was obtained by the combination of the wiki and gigaword corpora.

We also use different predictors including linear predictors, SVMs and Recursive Auto Encoders (RAE). It can be seen from 6 that complicated predictors do not improve the performance much. The best performance is achieved by using simple logistic regression. A detailed analysis is performed in VII. We also performed feature selection on the set of features. The final best system with the right set of features achieves a score of **0.51** $F1$. A complete error analysis is done in Section VII.

Our final best system uses the distributed representations from **wiki+giga** corpus, uses logistic regression predictor and uses the entire set of features. Table VI shows the performance of our best system compared the baseline and the oracle.

Method	Precision	Recall	F1	Accuracy
Baseline	0.70	0.38	0.50	0.72
Oracle	0.60	0.91	0.74	0.76
Best System	0.67	0.42	0.52	0.75

VII. Error Analysis

Traditional NLP tasks involved many hours of feature engineering and error analysis. The researcher had to look at the errors in his system and insert features to capture those errors. Although, this is true for all machine learning systems, this is specifically more true in the field of NLP where the data is unstructured. The infinite loop of feature engineering, error analysis and more feature engineering can be quite cumbersome. It does not end here, this kind of detailed feature engineering also leads to systems that do not generalize well. These systems over fit on the development set.

With the advent of distributed representations we gain much more information from these representations which otherwise needs multiple features and extensive tuning. Also, these vector representations provide much more generalized systems. In this work, we used vector representations trained on different corpora. We used code provided in [9] to train our glove model and code in [10] to train the RAE model. The other two models were trained using Google's toolkit [14].

From Figure 5 it can be seen that the vector representations from the provided google news data set is very close to the best system. But it does not provide the best performance. This is attributed to the fact that there is a mismatch between news data and standard twitter data. Although the vectors trained in the glove system are based on twitter data, the

objective of glove is to learn word vectors such that their dot product equals the logarithm of the word's probability of co-occurrence. This leads to good vector representations but for our task of paraphrasing entire sentences it does lend good phrase representations. The other two systems are both trained on the combination of wkipedia and gigaword corpus. This corpus has enough information to capture good representations. One of the two representations uses the skip gram method of training as described in [8], the other uses the continuous bag of words method of training. It can be seen that the skip gram performs slightly poorly when compared to the other method which can be attributed to the skip gram model not having enough context on either side to predict the current word.

In Figure 7 we perform forward feature selection, i.e, we add a feature at a time to determine the impact of each feature on the final performance. Added the L_2 similarity between the vectors gives us a high accuracy but very low precision, recall and $F1$. This is explained by the fact that the task has much more negatives as opposed to positives and it is the positives that we care about a lot since the evaluation metric is $F1$. After adding the cosine similarity between the combined vectors of the two phrases pushes the performance up drastically. This was expected since the vectors were trained to capture similarity in high dimensional spaces, hence, taking their cosine similarity will give us a measure of how much they mean the same thing. Adding NER features increases our recall a little bit. This feature tries to capture the overlap of NER tags between the two sentences. This can be useful when this effect is not captured well by the word vectors. Adding n-gram features increases the score by a significant amount since the cosine similarity cancels out same words between the two sentences, however, the presence of these words is a good indicator for paraphrasing. The n-gram features capture similarity where the word vectors fail to do so and hence these two features compliment each other.

Figure 6 shows the effect of different predictors on the performance. It can seen that all the predictors perform similarly and there isn't a clear winner. The similar performance of logistic regression and the SVM methods might be because the decision space might be clearly separable and hence SVMs don't provide any more benefits. The comparatively poor performance of RAE technique is because the technique considers the nodes of a parse tree and words and tries to learn representations for them. However, since our tweets are really short, the parse tree does not have enough nodes to provide the discrimination required. We decided to stick with logistic regression method since it was easier to train and debug.

We provide an example for each of the four classes here.

- **True Negative**
  A Walk to Remember is the definition of true love
  A Walk to Remember is on and Im in town and Im upset

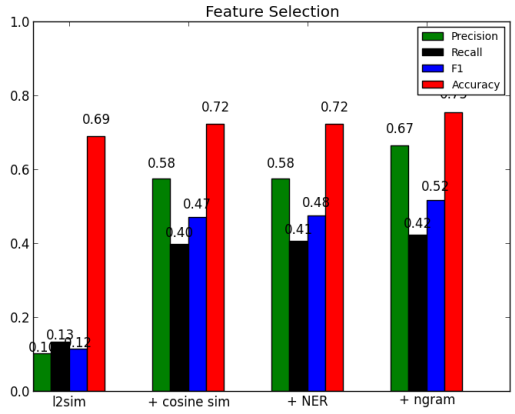

Fig. 7. Forward feature selection

- **True Positive**
  What the fuck did Amanda Bynes do to her face
  wow what the hell has happened to amanda bynes

- **False Positive**
  My phone is annoying me with these amber alerts
  what is this amber alert thing on my phone

- **False Negative**
  Rockets Asik showed the Thunder
  Omer Asik came through big tonight

It can be seen that in the first example, the system understands that the sentiment between the two sentences are different. In the second example, the vector representations of *what the hell happened* and *what the fuck did* are pretty close to each other and hence the system gets it right. The false positive looks really difficult and the system does not have enough discriminating features to get it right. In the last example, the system fails to learn that the phrase *showed the thunder* and *came through big* mean the same thing. This can be attributed to these phrases being rare in the training data.

VIII. Conclusion

We participated in the **SemEval2015** Paraphrasing in Twitter challenge. In this report, we present the challenges in processing small segments of unstructured text like tweets. We describe our data and corpora used. We give a brief primer on distributed word representations. We have tried four different methods which involved three types of training of word representations. Our final method was to use features from the best word representations in a classifier. We performed a detailed error analysis on the experiments with various representations, predictors and features. Our final best system achieves **0.52 F1** and **0.75 accuracy** which is a **4%**

and **4.2%** relative improvement over the baseline score. Our system achieves a high precision of **0.67** which is higher than the oracle's precision but a low recall. The final accuracy is better than the baseline system and is very close to the oracle.

IX. Future Work

The test data for the task will be released today (December 12th 2014). We plan to run our experiments on the test set and try do some error analysis. The evaluation for the task starts on December 18th. Our system will be an official submission from the Stanford NLP group along with Xiao Cheng. We hope we will do well in the task and will then be required to write a system description paper and present at the workshop.

X. Acknowledgment

We would like to thank Xiao Cheng, PhD student, Stanford NLP group for his guidance and for willing to collaborate with us. We would also like to thank Prof. Chris Manning for allowing this collaboration and his inputs on the problem. Finally, we thank Adam Goldberg our TA for his feedback at all the different stages.

References

[1] Regina Barzilay and Lillian Lee. Learning to paraphrase: an unsupervised approach using multiple-sequence alignment. In Proc. of NAACL, 2003
[2] Erwin Marsi and Emiel Krahmer. Explorations in sentence fusion. In Proc. of EWNLG, 2005
[3] Chris Callison-Burch, Philipp Koehn, and Miles Osborne. Improved statistical machine translation using paraphrases. In Proc. of HLT-NAACL., 2006
[4] Leon Derczynski, Allan Ritter, Sam Clark, Kalina Bontcheva, Twitter Part-of-Speech Tagging for All: Overcoming Sparse and Noisy Data. RANLP 2013.
[5] Brendan OConnor, Michel Krieger, David Ahn, Tweetmotif: Exploratory Search and Topic Summarization for Twitter. ICWSM 2010.
[6] Wei Xu, Alan Ritter, Chris Callison-Burch, William B. Dolan, Yangfeng Ji, Extracting Lexically Divergent Paraphrase from Twitter. 2014 Association for Computational Linguistics.
[7] Richard Socher, Eric H. Huang, Jeffrey Pennington , Andrew Y. Ng, Christopher D. Manning, Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection, NIPS 2011.
[8] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffry Dean Distributed Representations of Words and Phrases and their Compositionality, 2013
[9] Jeffrey Pennington, Richard Socher, Christopher D. Manning GloVe: Global Vectors for Word Representation
[10] Richard Socher and Christopher Manning Deep Learning for NLP NAACL 2013, Atlanta.
[11] Alex J. Smola, Bernhard Schölkopf, Klaus-Robert Müller The connection between regularization operators and support vector kernels 22 December 1997.
[12] English Gigaword Fifth Edition Online Dataset description https://catalog.ldc.upenn.edu/LDC2011T07.
[13] Dipanjan Das and Noah A. Smith Paraphrase Identification as Probabilistic Quasi-Synchronous Recognition 2009 ACL and AFNLP.
[14] Google word2Vec training tool kit. code.google.com/p/word2vec/.
[15] https://github.com/sem-io/python-twitter-spell-checking