Deception Detection

Zhongyang Fu(zf52), Yan Huang(yh553), Ke Wang(kw427), Rakesh Recharla chenchu(rr548),
Vinayaka Dattatraya(fv59), SaiKiran Mullaguri(sm947)

1. Features Generation:

Here, we developed features based on the following methods:
a. Word n-gram
b. Character n-gram
c. Parts of Speech tags
d. TF-IDF

a. Word n-gram:

The given training, validation and test data is preprocessed by using stemming and by removing stop words in the data. We removed stop words from the given data because these words appear most frequently but have nothing to do with the deception detection.
We have created a python script "first.py" which removes all the stop words in the training set. The output of this python script is provided to python programs "unigrams.py", "partsofspeechunigrams.py", "char-trigrams.py" which gives corresponding unigrams, parts of speech tags and char-tri grams. To generate features for SVM for the output files which are emitted by above mentioned python programs, we have implemented two hash tables in java program
1. "featureMap" which stores key,value pair where key is the string and value is the unique feature number.
2. "sentenceMap" which is a sorted hashmap of key,value pairs where key is the unique feature number and value is the count of that word in the particular sentence.
These two hashmaps are used to generate the feature set file required for svm_learn in the format as shown below. For example,

"This is a very very good idea"

The feature value pairs to be fed to svm_learn are generated as follows

truthful	deceptive	value
0	0	-2
0	1	-1
1	0	1
1	1	2

-1 1:1 2:1 3:1 4:2 5:1 6:1

Here, -1 is review value generated tag, x:y represents <feature value>:<count of the occurrence of the word in a review>. Note that the tag and sentence level content is stored in the feature set file. The generated features are then fed as input to SVM.

b. Character n-gram:

Here, the above mentioned procedure is followed to generate character-wise trigrams by using a code "vin=ngrams(chrs,3)" imported from nltk. Values in the function can be changed to

generate character-bigrams, character-trigrams etc., The effectiveness of using character n-grams is described in experimentations section.
we have implemented trigram for character n-gram. Here we have used 3 letters for generation.

c. Parts of Speech:
Unigrams obtained in training, test and validation data are tagged using nltk tagger and we generated features using the similar approach as mentioned above.

d. TF-IDF:
We use TF-IDF to identify the words which have more value for our task, and use those words as our feature set. We consider every sentence(Si) as a document, and those have the same deception label(D(Si) = 1 or -1) as they are in the same group. In this way, we can give a TF-IDF score to every word(Wij) in the sentence(TFij). Then we define the score for each word:

Score(Wij) = Sum(D(Si)*TFij) for every W = Wij

After having a score, we use a threshold to filter out those we think is not useful for us. We tried different thresholds and finds out that about 100 features is the best fit for this task.

2. Analysis of previous phases:
a. Phase 1 Analysis:
We executed algorithm on test data and the classifier classifies whether the review is deceptive or not. Most rules that humans use to tell the deceptive reviews that we found in phase 1, match the result. (e.g too many focus on the family member like “my husband and I went to...”). Since, human can not do statistics as fast as a computer does, the algorithm developed would run more accurately and at a much faster rate.

b. Phase 2 Analysis(Comparison with Gold-Standard):
We thought that the review that has strong recommendations and over-praising the hotel in each and every line as deceptive and marked zero for that.
Considering the example(118th review)
1. “Hotel Monaco Chicago - a Kimpton Hotel;Upon entering the Hotel Monaco Chicago; i could see the beauty of a classically.....I would recommend this hotel not only for the pure elegance; but for the wonderful customer service and the fantastic amenities”, we marked it as deceptive but gold standard suggests it as truthful. Since usage of more adverbs and recommendations in the review, We thought it as negative, discussed with team members and we all agreed to that but it got wrong. The reason might be we didn’t consider word “skeptical” in the review to classify it in the correct sense. So, we have to consider negative words into account in addition to adverbs features as well.
2. We classified reviews that have details like” deals ending up soon on the website” as truthful and is correctly reflected in the gold standard as well.
3. We considered strong recommendations as deceptive but according to gold standard, giving weightage to words that are negative in the review besides recommendations might be the correct approach to predict whether it is deceptive or truthful.
For example(96th review),“Hotel Monaco Chicago - a Kimpton Hotel;We loved this hotel and

would definitely come back......we weren't given a goldfish; which was disappointing...”
Here in the above example, “disappointing” is given more weightage than recommendations in the gold standard.
We took time to rate reviews based on features obtained from phase 1, annotated individually and for each review, based on the majority of ratings, we have given the overall rating for the review. In the case of conflicts, we cross-checked once again with the features we generated for phase 1 and rated them accordingly.

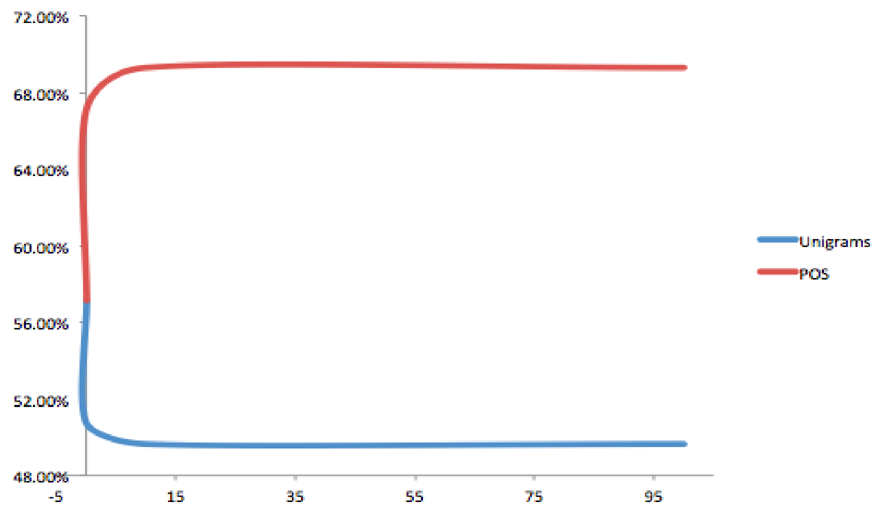
c. Phase 3 Analysis:
We analysed gold standard with that of the annotated reviews, taken observations into account and wrote deceptive positive and negative reviews for this phase.

3. Algorithms of Phase 4:
1) SVM:
We use the given **SVM-*light*** to train our data.
First, We must preprocess the data file since the label for each record is <*isTruthful, isPositive*>. We used ***split.py*** and ***rename_label.py*** since we only need the label of “is*Truthful*” in our algorithm. It must be 1 or -1 to meet the format requirements of SVM-*light*.
After we get the processed data, We used SVM-*light* to learn, validate and test the data. I tried unigrams, bigrams, pos, char-unigrams in this algorithm.
We did cross-validation to choose the best parameter C (leave-one-out) : Considering **TF-IDF** for example, the results are as follows: (More information are in the data file)

C	estimate of the error	estimate of the recall	estimate of the precision
0.001	31.67%	42.12%	72.50%
0.1	28.29%	60.37%	69.62%
10	28.29%	60.37%	69.62%
100	28.29%	60.37%	69.62%
1000	28.29%	60.37%	69.62%

I have chosen the best C, and tested on the test data, the accuracy is about 0.745.

<u>**Plot and Observations:**</u>
The figure below shows the curve of performance on validation data and parameter C of Unigrams and POS feature. **X-axis** represents the value of parameter C as 0.001, 0.01, 0.1, 10, 100... and **Y-axis** represents the performance(accuracy) on validation set. From the figure, the best value for C is around 0.1.



2) Language Model:

We build our language model using unigram and bigram data. We first split the training data into two different segments-- truthful review and deceptive review. Afterwards, we trained two models using different reviews. Then, for each piece of review that we have, we generate the unigrams and bigrams from the input file, and calculate the probability of being truthful or deceptive using the following equation for unigram models:

$$P(t_1 t_2 t_3) = P(t_1)P(t_2|t_1)P(t_3|t_1 t_2) \text{ to } P_{uni}(t_1 t_2 t_3) = P(t_1)P(t_2)P(t_3)$$

For bigram models:

$$P(W_n|W_{n-1}) = \frac{P(W_{n-1}, W_n)}{P(W_{n-1})}$$

Lastly, we add the log base 10 value of each probability together, and choose the model with the highest probabilities sum.

Unknown words: For any words which are not exist in the training data, we set the probability as really small: 0.00001 compared with all the other existing probabilities.

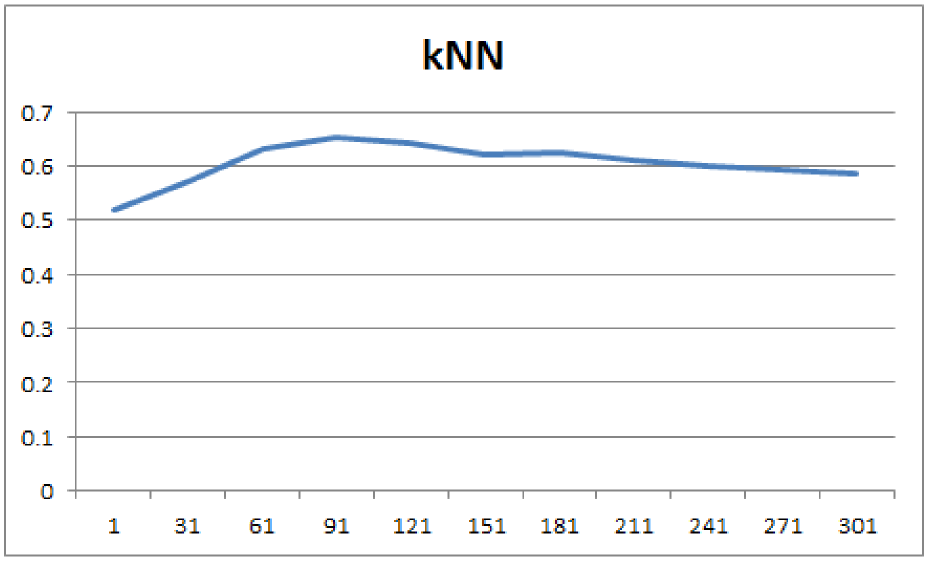
Validation set: For unigram model, we get 203 right guess out of 280
For bigram model, we get 220 right guess out of 280

Results: For unigram model, we get a kaggle score of 0.80
For bigram model, we get a kaggle score of 0.711

3) kNN:

We tried different k values for the kNN algorithm, using validation set. The result is showed in the following chart.

<u>Plot and Observations:</u>



As we can see, the best value for k is 91. It's almost a random guess when k equals to 1. The overall performance is not great in this experience. One reason may be that we only use TF-IDF features in this experience.

4. Extensions:

1) Additional Feature - TFIDF:

As described above.

2) Additional Algorithm - Language Model:

As described above.

3). Using polarity as a feature in deception detection:

a) Implementation:

We used SVM to cascade two classifiers.

Firstly, used ***split.py*** to preprocess the bigram data, that split them into two different files, one is truthful/deceptive label and the other is positive/negative label (which are relabeled to meet the requirements of SVM-*light*).

Using SVM-*light* to classify whether the review is positive or negative using the positive/negative version of training data. ***getLabel2.py*** assign a polarity to each review and store the result into two different files: one is positive reviews, the other is negative.

Secondly, use ***split2.py*** to deal with the original data, put the positve reviews into a file (e.g. unigramsTrain_positive), and negative reviews into another (e.g. unigramsTrain_negative).

Using SVM-*light* again to classify whether the review is truthful or deceptive. I train and test to positive reviews and negative reviews separately. After we get the truthful/negative label, run ***merge.py*** to merge the positive and negative reviews.

b) Evaluation:

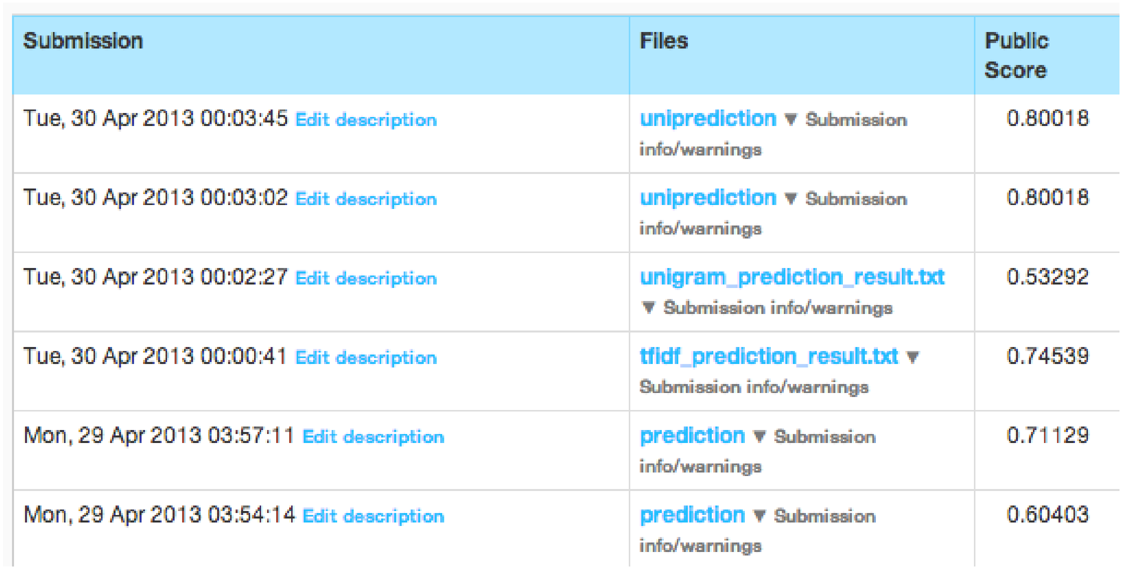
The accuracy is not so good as expected. The reason may be that we do two rounds of classification of SVM. Each time we need to tune the parameter C, which can generate prediction error. This inaccuracy would affect the classification in the second round, which would enlarge the error rate.

5. Experiments and Result analysis:

a. Research paper on "Deception Detection" performs text categorization task using unstemmed words. We experimented by using stemming beforehand to generate accurate features.
b. Results prove that text categorization using **Unigrams** gives an accuracy of 80 percent in correctly predicting deceptive or truthful reviews than **Bigrams**(71.1 percent) and **TF-IDF**.
c. We find that most of the observed features in deceptive and truthful reviews satisfy table of most weighted words mentioned in the research paper which concretely proves the correctness of text categorization approach.
d. We performed folded cross-validation by keeping 25 percent of train data as test data, executed algorithm on remaining portion of training data before testing that on validation and test datasets.

6. Kaggle Competition Results:

kaggle team name: **yh553_kw427_fv59_rr548_sm947_zf52**



Submission	Files	Public Score
Tue, 30 Apr 2013 00:03:45 Edit description	uniprediction ▼ Submission info/warnings	0.80018
Tue, 30 Apr 2013 00:03:02 Edit description	uniprediction ▼ Submission info/warnings	0.80018
Tue, 30 Apr 2013 00:02:27 Edit description	unigram_prediction_result.txt ▼ Submission info/warnings	0.53292
Tue, 30 Apr 2013 00:00:41 Edit description	tfidf_prediction_result.txt ▼ Submission info/warnings	0.74539
Mon, 29 Apr 2013 03:57:11 Edit description	prediction ▼ Submission info/warnings	0.71129
Mon, 29 Apr 2013 03:54:14 Edit description	prediction ▼ Submission info/warnings	0.60403

7. Individual member contribution:

Zhongyang Fu(zf52)	Implementation of TF-IDF feature and KNN algorithm
Yan Huang(yh553)	Implementation of language models
Ke Wang(kw427)	Implementation of SVM and extension of polarity
Rakesh Recharla chenchu(rr548)	Data preprocessing and bigrams generation
Vinayaka Dattatraya(fv59)	Analysis of previous phase results.
SaiKiran Mullaguri(sm947)	Generation of unigrams, char-trigrams and parts of speech features