Project Definition

Proposal

Movie review classification with machine learning techniques into two categories - positive and negative reviews

Overview

With the amount of opinions expressed on social media, a sense of public opinions can be assessed if the “polarity” of opinions on a specific topic can be assessed. An accurate assessment of public opinion is a very useful tool for marketing.

Unfortunately, this is not a trivial tasks given the scale of data available and variation in the sentence construction. Since human languages analysis with hard coded rules will be brittle, machine learning is the natural answer. They can be potentially trained to be able to classify the statements according to the inferred sentiment.

Sentiment analysis is a challenging subject in machine learning. People express their emotions in language that is often obscured by sarcasm, ambiguity, and plays on words, all of which could be very misleading for both humans and computers.

I work in semantics development for a graphical modelling language in my day job. This has increased my interest in natural language processing (and semantic inference in particular). I used a kaggle competition https://www.kaggle.com/c/word2vec-nlp-tutorial as reference for this.

The introductory section gave the traditional approach to Semantic classification of sentences and the Deep Learning inspired approach to understanding word semantics. I also used the following as reference for understanding about the concept of paragraph vector for taking word ordering also into context http://cs.stanford.edu/~quocle/paragraph_vector.pdf
https://districtdatalabs.silvrback.com/modern-methods-for-sentiment-analysis .

Apart from word vectors, recurrent neural networks also seem a natural fit for NLP. Unlike images, sentences can be of varying length. And the ordering of words affects the meaning of sentence to a very large degree. Therefore, I also studied RNN performance with binary sentence classification. I used the following links for reference
http://deeplearning.net/tutorial/lstm.html
http://karpathy.github.io/2015/05/21/rnn-effectiveness/
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Problem Statement

The primary objective of my capstone project is to classify movie reviews into two classes - positive or negative. I will use a few different techniques and comparing their results.

For this I will use Kaggle project https://www.kaggle.com/c/word2vec-nlp-tutorial as a reference and data source. I will also explore some other deep learning approaches I saw in http://deeplearning.net/tutorial/lstm.html .

Metrics

Area under receiver operating characteristic (**ROC**) curve of the algorithm.
Accuracy is measured by the area under the ROC curve. An area of 1 represents a perfect test. On the other hand, an area of .5 represents a worthless test.

ROC curve would plot the False positive rate (x-axis) v/s True positive rate (y-axis). Intuitively speaking, greater area under curve would mean higher true positives to false positive ratio.

Analysis

Data Exploration

I shall draw the dataset(s) from https://www.kaggle.com/c/word2vec-nlp-tutorial/data

The labeled data set consists of 50,000 IMDB movie reviews, specially selected for sentiment analysis. The sentiment of reviews is binary, meaning the IMDB rating < 5 results in a sentiment score of 0, and rating >=7 have a sentiment score of 1.

No individual movie has more than 30 reviews. The 25,000 review labeled training set does not include any of the same movies as the 25,000 review test set. In addition, there are another 50,000 IMDB reviews provided without any rating labels.

The input data consists of the following data repository. There is an option to try unsupervised learning technique also and therefore there are both labeled and unlabeled data sets.

- **Labeled Train Data** - The labeled training set. The file is tab-delimited and has a header row followed by 25,000 rows containing an id, sentiment, and text for each review.
- **Test Data** - The test set for Kaggle competition. The tab-delimited file has a header row followed by 25,000 rows containing an id and text for each review. Your task is to predict the sentiment for each one. Although this is not useful to the Supervised Learning

approaches considered, it can be useful for training the unsupervised part of the approaches.
- **Unlabeled Train Data** - An extra training set with no labels. The tab-delimited file has a header row followed by 50,000 rows containing an id and text for each review. Like the Test Data, this shall be useful only for the unsupervised part of the approaches considered.

Since the data is all imported from HTML page, the preprocessing step will need to clean up the data.

Algorithms and Techniques

I implemented a traditional bag of words based classifier for sentiment analysis to set the benchmark for this. I used different recent approaches for sentiment analysis:

1. Word vector with averaging (weighted and otherwise)
2. Paragraph vector
3. Recursive Neural Network
4. Ensemble Learning (Majority vote)

I measured the performance over the evaluation metric (ROC curve area) on test data. In addition to that there will also be the additional challenge of whether the data will need to be fully supervised or not (paragraph vector does not need to be fully supervised as per the quoted literature and needs no parsing).

Bag of Words

The traditional approach for sentiment analysis ignores the order of words in a sentence and uses the count of frequent words. Each unique word is considered a separate dimension of solution space (one of K encoding).
It accumulates vocabulary as it trains on more data.
Sentence 1: "The cat sat on the hat"
Sentence 2: "The dog ate the cat and the hat"
From these two sentences, our vocabulary is as follows:
{ the, cat, sat, on, hat, dog, ate, and }

And the features for the two sentences is :
Sentence 1: { 2, 1, 1, 1, 1, 0, 0, 0 }
Similarly, the features for Sentence 2 are: { 3, 1, 0, 0, 1, 1, 1, 1}

The feature value helps us identify sentiment of the sentence in bag of words. And it is non-intuitively very successful method.

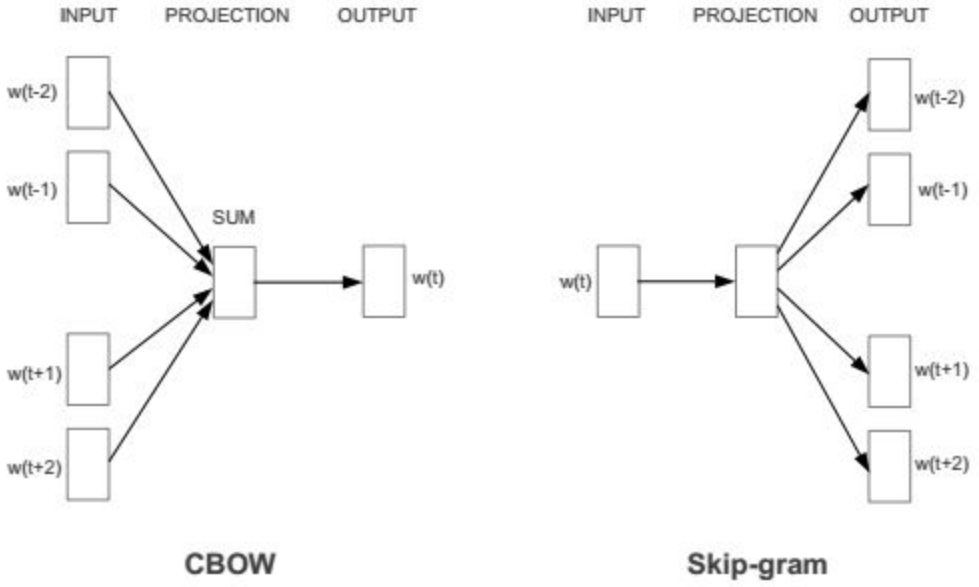
Beyond the counting and the feature value assignment, we can use any supervised classifier. I chose Random Forest for my classification.

One thing to note is that this works best for long enough sentences. But it does lose crucial semantic information by ignoring sentence structure. For example, “I am not not interested” shall get classified as a negative sentiment since all the classifier will look at is [(“I”, 1), (“am”, 1), (“not”, 2), (“interested”,1)]. By using the anchor words “not”, “interested” it will classify it as negative sentiment.

Word Vector

Word Vector simply put can be thought of using neural network to embed words using a non-linear transform into a different space which reduces the size of data and reduce the context of the word. I used the gensim libraries to generate word vector from the IMDB training set provided. The benefit of using word vector as the first stage of classification is that it is an unsupervised training technique (which is how most of the data is real world). Therefore, I could use test data and unlabeled training data for word vector model training as well.

Gensim libraries that I used implemented word vector model from https://arxiv.org/pdf/1301.3781.pdf. They are inspired by Neural Networks but use a different approach. There are two model architectures proposed- Continuous bag of words (CBOW) and Skip-Gram. There are other architectures for word vector calculation (including using Neural Networks) but the paper proposed these to be computationally less expensive.



Continuous Bag of Words

This architecture uses 2 steps approach (without the hidden layer of a neural network). It simply uses the projection layer for all words such that the ordering of words does not matter as this

would be repeated for every word in the context (hence the bag of words in the name). However unlike Bag of words it uses continuous distributed representation of context.
The paper proposes using a log linear classifier with training criterion to classify the current word.

Continuous Skip Gram
Similar to CBOW, this tries to assign a word vector to a word on the basis of the words in context. Each word is used as input to a log linear classifier with continuous projection layer. It then predicts the words in range before and after the word. However, the more distant words can be treated differently in this architecture (e.g. by sampling them less).

These word vectors now capture the context of surrounding words. This can be seen by using basic algebra to find word relations (i.e. word_vector("king") – word_vector("man") + word_vector("woman") = word_vector("queen")). These word vectors can be fed into a classification algorithm, as opposed to bag-of-words, to predict sentiment. It has two main advantages

1. Our feature space is much lower (typically ~300 as opposed to ~100,000, which is the size of our vocabulary).
2. Very little manual feature creation since the model extracts those features for us.

I tried to use the output of Word Vector learning in the following classifier approaches

Weighted Averaging

Since sentences can have varying lengths, using an average of word vectors allows a common metric to compare with other word vectors .

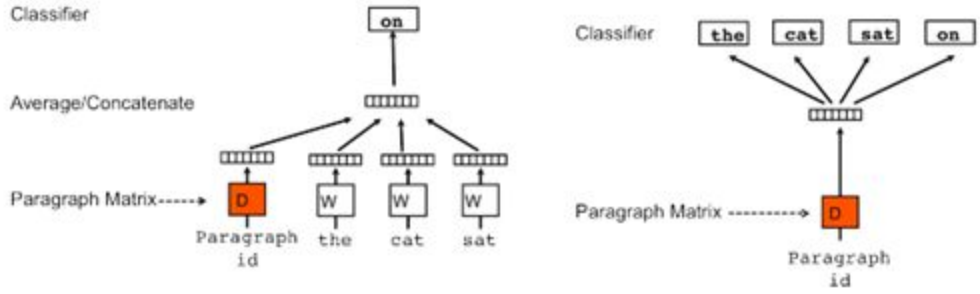
Bag of Centroids

Using a similar logic, bag of centroids is another approach that can be used to bring variable length sentences to a common ground. This approach would involve finding centroid of word vectors in the sentence (or paragraph) and use that for classification.

While both of the above approaches get the advantage of reduced feature space and avoid manual feature creation, they both ignore the sentence structure and all associated semantics. Therefore, they both don't do much better than bag of words. I verified this in my experiments as well.

Paragraph Vector

To generalize the concepts of word vectors to paragraphs/sentences of varying lengths, the concept of paragraph vector was introduced in
http://cs.stanford.edu/~quocle/paragraph_vector.pdf



The main idea is to implicitly capture the paragraph semantics by defining a paragraph vector for each paragraph similar to word vector for each word. In the architectures proposed, the word vectors are learnt before the paragraph vectors are identified. While the word vectors are shared across paragraphs, the paragraph vectors are not
Analogous to Word Vector, there are two methods

Distributed Memory (DM)

Every paragraph is mapped to a unique vector, represented by a column matrix D.
DM attempts to predict a word given its previous words and a paragraph vector. Even though the context window moves across the text, the paragraph vector does not (hence distributed memory) and allows for some word-order to be captured.
The algorithm has 2 stages

1. Training to get word vectors, paragraph vectors on already seen paragraphs
2. “Inference stage” - get paragraph vectors for new paragraphs by adding more column to D (Pargraph matrix) and using gradient descent while holding word vector and other parameters constant.

Distributed Bag of Words (DBOW)

This ignores the order of words and uses an approach to consider words randomly sampled. It tries to predict words randomly sampled from the paragraphs in the output.

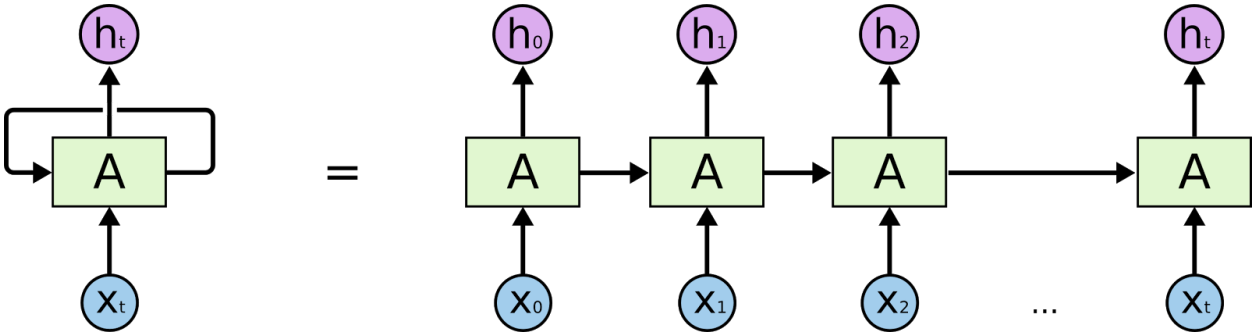
That is, for each iteration of gradient descent a text window is sampled, then a random word is sampled and a classification task given the paragraph vector.
DBOW predicts a random group of words in a paragraph given only its paragraph vector .

Once it has been trained, these paragraph vectors can be fed into a sentiment classifier without the need to aggregate words. One very big advantage it has is that it is unsupervised learning technique and thus can be trained over a much larger dataset.

Recurrent Neural Networks (RNN)

Text is usually of varying lengths which can make windowing a challenging task in Feedforward neural network. RNN are a natural answer for neural network to use for

NLP.(http://colah.github.io/posts/2015-08-Understanding-LSTMs) RNNs give an efficient way to represent sequences in general. They can be unrolled to represent a feed forward neural network.



Recurrent neural network behave akin to a computer program with memory. Since words have associations across sentences and these associations can change semantic meanings, RNN help resolve the issue.

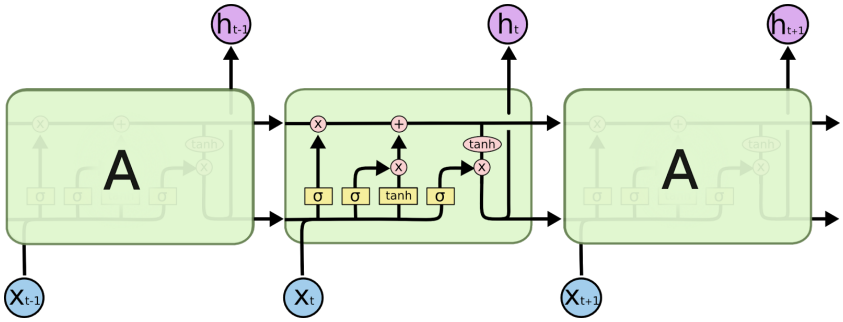
However, practically they have the challenge of exploding and vanishing gradients. For example:

(learning_gradient) ^k ->0 as k->inf if learning_gradient<1

Or

(learning_gradient) ^k ->inf as k->inf if learning_gradient>1

Both these issues don't allow RNN to perform well with sequences beyond a certain length.To address this LSTM cell was introduced (http://colah.github.io/posts/2015-08-Understanding-LSTMs) which replaces regular cells as shown in the following figure



The gates in a LSTM cell optionally passes through signals with the use of its gates. Unlike a typical feed forward neural network, this is important in RNN as it has backpropagation through time and the number of times it has gone through cells becomes too large and it is unable to connect points in the sequence.

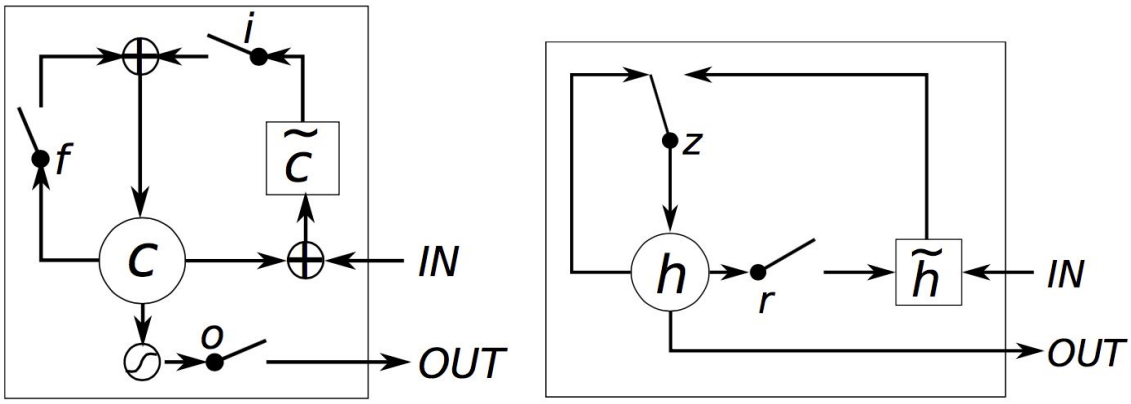


Figure: LSTM cell v/s GRU Cell

GRU is a lighter weight version of LSTM as it has lesser number of parameters that require tuning. However, it has the same performance as LSTM cell(https://arxiv.org/pdf/1412.3555.pdf).
One downside of RNN is that it takes a lot of data to be trained over.

Ensemble Learning

Ensemble learning is an approach to combine results of multiple models and try to get best of all worlds. While there are many flavors of ensemble learning, since I already had results from all different models - I used bootstrap aggregation (bagging) to evaluate the performance.

While in small runs, it will be no better than best model, in long run - it can do better than any model in the set.

The solution therefore will be to identify the optimal approach among the algorithms identified to use for classifying movie reviews.

Benchmark

The traditional bag of words combined with a Random Forest shall serve as the benchmark model. The area under ROC curve will serve as a metric to compare with the other approaches. As shown in the ipython notebook, the ROC achieved = 0.8454.

Methodology

Data Preprocessing

As mentioned in Data description, the movie reviews have been mined from HTML pages. So each review contains HTML tags irrelevant to sentiment classification and therefore need to be removed.
It did not make sense to write regular expression search to replace tags as that would be brittle and not tested enough. So, I used Beautiful Soup library to help with cleanup (https://www.kaggle.com/c/word2vec-nlp-tutorial#part-1-for-beginners-bag-of-words). For each input string, I need to create a BeautifulSoup object from the string and calling get_text() method on it, returns a string without the tags and markup.
Apart from removing tags, there can be expressions using punctuations and numbers. While these can be part of an expression, they do not constitute a word strictly. Punctuations can sometimes change the meaning of a sentence entirely. For example, “Most of the time travellers worry about their baggage” v/s “Most of the time, travellers worry about their baggage”.

However, in certain approaches(like Bag of Words), it makes sense to remove the punctuations and reduce features considered as the order of the words does not matter. Similarly, some punctuations can form emojis like “:)”, “:(”, etc. However, for the sake of simplicity they were removed. Similar arguments would apply to usage of numbers in sentences.

Apart from numbers and punctuation a similar argument would apply to “stop words” i.e. words with little meaning. For example, “a”, “the”, “and”, “is”, etc. We can identify and remove them using Python NLTK.

I used the approach mentioned in (https://www.kaggle.com/c/word2vec-nlp-tutorial) for preprocessing. It used Python regular expressions combined with NLTK for tokenization and removing “undesirable or non words” conditionally. It returns list of validated words from a review in lower case (to reduce the number of features to deal with due to variation in capitalization).

Ideally I would’ve liked to minimize language specific (here English) preprocessing. However, given that I had limited computational resources at hand and all the reviews are in English, it made sense to use this for the preprocessing.

Implementation

A lot of help was available for the libraries I ended up using. To make an accurate assessment of each method's performance, I split the labeled training data in the start itself keepnig 0.33 x (training data) for validation. This split of data was used for all approaches.
Since there were some unsupervised learning algorithms also used in some approaches, I kept the unlabeled training data also around.
As some methods required the data pre-processing different from other - I kept the pre-processing of data as utility functions rather than pass processed training and validation data.
Since some of the algorithms took really long to train (sometimes better part of the day on my "not so ancient but a bit dated" laptop), I logged all the test data in Code/logs/* path. Also, I kept all the data under Code/data/* path

Bag of words

Bag of words used CountVectorizer with max_features of 5000 which would be the vocabulary size. The code ran reasonably fast (especially compared to RNN algorithm) and using a scikit RandomForest, I was able to implement the method. It was simple from coding perspective and fast in execution. I used the code accompanying Kaggle competition for getting started with this method.

Word vector based methods

Word vector method needed the whole vocabulary to be fed to the word vector model. I used gensim library for this. After training, I saved it to Code/data/<word vector model name> to avoid repeated generation for different approaches taken. I selected the parameters using gensim documentation and https://arxiv.org/pdf/1301.3781.pdf .The API for gensim word vec model was relatively straightforward. I used the code accompanying Kaggle competition for getting started with this method.

Averaging Vectors

After word vector model training, I used a random forest (with same depth as Bag of Words Random Forest) over averaged word vectors for each review. To get word vectors, I used index2word feature to identify word vector for each word in review

Bag of Centroids

I simply used k-means cluster (num_cluster = 5, a default suggested in Kaggle competition documentation) method to identify clusters. I then used random forests to identify where a certain review lay according to centroid it was associated with.

Paragraph Vector

I used a similar approach as word vector. I used gensim Doc2Vec to implement the paragraph vector estimation followed by classification using SVC Classifier. Doc2Vec takes the input in form of LabeledSentences to capture words with a unique label for each review.
Using the gensim FAQ and documentation as reference, I fed the data in 10 epochs after shuffling the sentences fed to the model. I trained both distributed memory and distributed bag of words with the same features to draw a fair comparison between the two methods.
After training the two models I saved them to Code/data/ <model name> "_dm" or "_dbow" for future reuse.
I used the output of the Paragraph vectors in SVC classifier with default attributes.

Recursive Neural Network

I initially spent a lot of time trying to get my code for Tensorflow to work for RNN (variable length). However, since most of the effort was directed at learning more about the tool rather than technique, I decided to use an easier to use package for the purposes of this project. Therefore I used the approach one of the participants in the Kaggle competition had posted to use https://github.com/IndicoDataSolutions/Passage.
I used a very high number of features for the tokenizer as it used a one-of-K encoding.
For the dataset I had and slow machine, I chose Adadelta as the optimization method. I used GRU instead of LSTM to speed up the training process (fewer parameters to tune).

Used a dropout to avoid co-adaptation of features. Since it is in hidden layer, we can choose a value high enough (I used 0.75) to avoid overfitting (https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf). The rest of the input attributes are the same as default attributes. I used sigmoid for 1 cell in Dense layer as it is a classification problem and I wanted it to be probability estimate for positive sentiment classification.
I use 0.5 as threshold to decide if the validation set input should be classified as positive review.

Ensemble Learning

I used a simple bagging approach for evaluating the result when using majority voting technique. It would ideally reduce the chance of overfitting and in large test set show an improvement over all the underlying models.

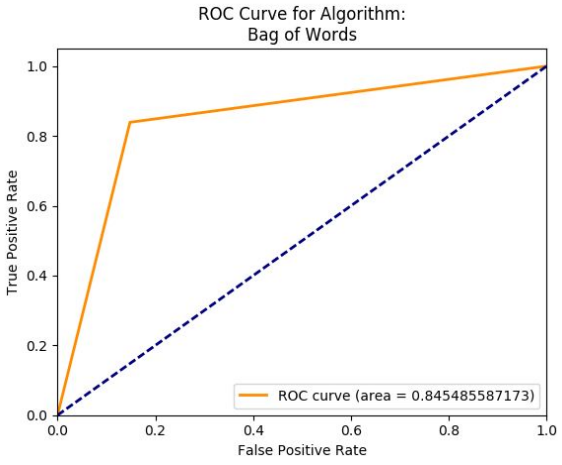
I used a simple averaging approach of logged results of all other methods. It assumes all the CSV files under Code/log/ path are output logs from other methods.

Therefore it is important to clear the CSV file logged for Boosting method.

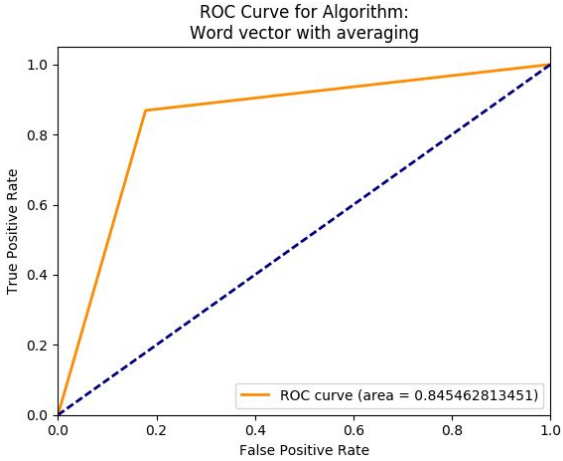
Results

Visualization

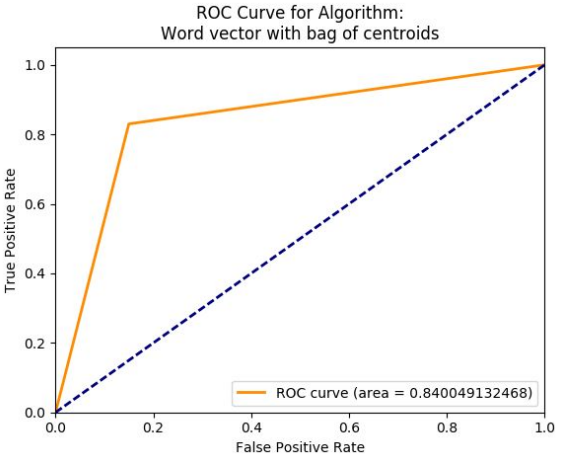
I used the results logged for each algorithm in the iPython notebook to create ROC curves for each using the code in submission “Code/plotGraphs.py”



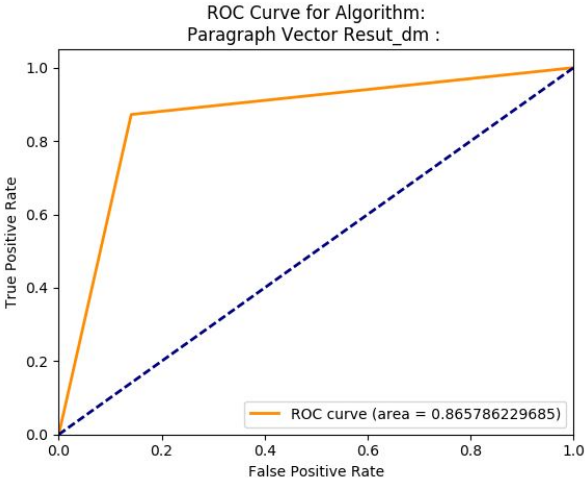
a)



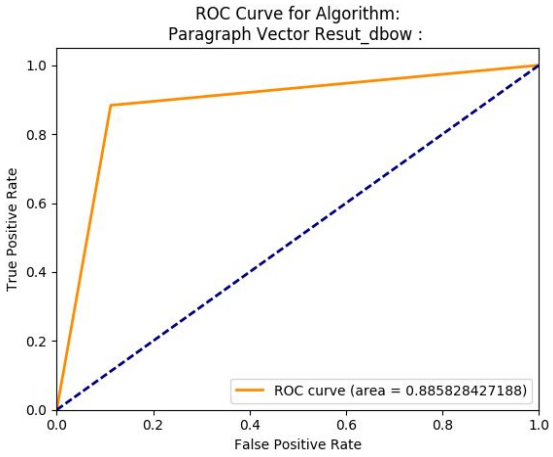
b)



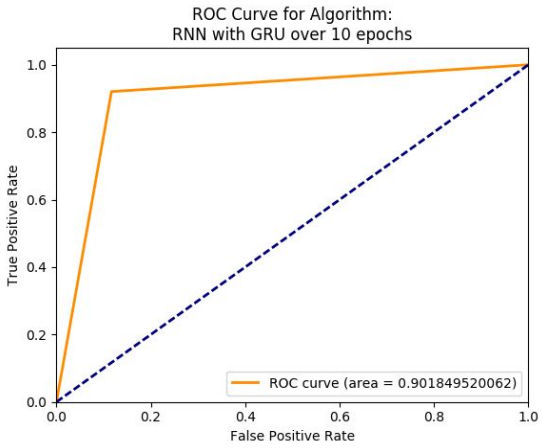
c)



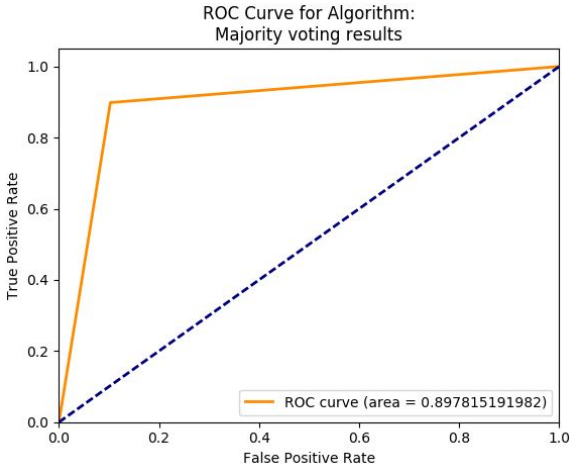
d)



e)



f)



g)

Figure 1. ROC Curves for different Machine Learning algorithm tried. a)Traditional approach using bag of words. b) & c) Word Vector library used with word vector averaging and K-means clustering. d) & e) Paragraph vector based classification (distributed memory and distributed bag of words) f) RNN (using GRU) based classification g) Majority voting ensemble over the models used

Model Evaluation and Validation

ROC under curve for (approximated to 5 decimal places):

1. Word Vector with Averaging = 0.84546
2. Word Vector with bag of centroids = 0.84005
3. Paragraph vector distributed model = 0.86579
4. Paragraph vector distributed bag of words = 0.88582
5. RNN (GRU based and on of K embedding) = 0.90185
6. Ensemble (Majority voting) = 0.89782

Of the multiple approaches tried, Paragraph vector (distributed bag of words and distributed memory), RNN and majority voting ensemble are the only ones outperforming Bag of Words (0.8454). Surprisingly , Distributed Model is worse off than Distributed Bag of Words for Paragraph vector. One reason can be the training set here is smaller so the advantage of implicitly maintaining syntactic structure of sentence isn't as evident.

Ideally, ensemble approach should be able to outperform most of the methods. However, we do include output of all methods in current approach. If limited to higher performing methods only, it should perform better. Also a scheme based on weighting of approaches shall intuitively perform better.

Time taken by all training methods apart from Bag of Words was on the higher side. The case of RNN was especially bad in terms of time taken. That does cut some points from making it method of choice.

Conclusion

Reflection

It was interesting to work on a project where the training time of model was high enough for a regular machine to take more than a few hours on a typical laptop.
Even though Bag of Words has quick training time, the loss of semantic association (of words and paragraphs) and structure of sentence does catch up in performance. This is the primary reason Paragraph Vector approach with moderate number of features (lower than Bag of Words) is able to outperform bag of words.
As computation power is getting inexpensive (especially with GPUs), it makes sense to move towards more intensive approaches as looked at here.
However, RNN was still on the slower side for all the models. A big part of the reason may be that it used bag of words approach to create one of K encoding and build vocabulary. This is where word vectors can really help RNN a lot by reducing the size of each tensor. This should help reduce computations thereby reducing training time significantly (100000/300 = 333.33 times at least).
When I tried increasing vector dimensions used for word vector model and paragraph vector model, I saw a small incremental improvement. That could be something to keep in mind, but it may need to be considered in addition with computational power at disposal)

Improvement

Given the points noted in the conclusion, I have identified following improvements:

1. Try pre-trained word vector models and paragraph vector models. These should be able to perform better with our training, test datasets

2. Word vector embedding for RNN input instead of one-of-K encoding. This should really cut down the computational cost. It will also have implicit closeness and directionality for certain words. For example, ‘big’ is to ‘bigger’ as ‘small’ is to ‘smaller’. This relation should be represented by word vector somehow
3. An obvious thing to try would be to try to combine with larger datasets (like rotten tomatoes dataset). I would be interested to see if distributed memory outperforms distributed bag of words for paragraph vector models.
4. Better ensemble approach (stacking or bucket of models) should be able to improve the results beyond any individual model. Also making it selective to work with only highest performance models should give better results.