Sentiment Analaysis

Akshat Jindal

March 25, 2018

1 Introduction

For the task of sentiment analysis, which is in essence a binary classification problem, we first want to get vector representations of the documents.Once this vector representation has been created, our problem boils down to learning a binary classifier. Here, we explore various vector representations and then try each of them on various different classifiers to get some interesting insight. Two code files :

1. GetVec.py : Main code file. Run like : $python GetVec.py
2. doc2vec.py : Code to train paragraph vectors.Run like : $python doc2vec.py

2 Dataset

We use, **Stanford large movie review** dataset available at `https://ai.stanford.edu/~amaas/data/sentiment/`. The core dataset contains 50,000 reviews split evenly into 25,000 train and 25,000 test sets.

3 Document Vector Representations

For this assignment, we must keep in mind that 1 document is actually 1 review. Also, nearly all methods require us to clean the data, i.e tokenize it,remove stopwords,perform stemming etc. For all this, the nltk library has been used.

1. **Bag of Words** : In this approach, each review is treated as a binary bag of words.Thus, a vocabulary is first created, by using the given vocabulary file **imdb.vocab**, after doing stopword removal. Now, a $|V| * 1$ sized binary vector is created for each review, with ones at the words it does contain, zeros at others. The CountVectorizer library from sklearn has been used to create these vectors.
2. **Tf-Idf Representation** : This approach is absolutely similar to binary bag of words,with the only difference being that instead of creating binary vectors of 1s and 0s, the $|V| * 1$ sized vectors we create for each review

have the tf-idf of the corresponding word. While the term frequency,being a local feature is calculated by simply counting the number of times the term occurs in the document(and then normalising it),the idf is a global feature and has to be calculated by looking at the entire corpus. Yet again, we use the TfidfVectorizer library from sklearn to create these vectors.

3. **Word2Vec**: In this approach, we first obtain the word vectors for each word in the review. Then these vectors are averaged to get the vector for the review as a whole. Two approaches were tried :

   (a) Simple averaging

   (b) Weighted averaging, with the tf-idf of the terms being the weights

   For obtaining the word vectors for each word,we use pre-trained vectors trained on part of Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases. We are extracting the word embeddings of words that are present in our vocabulary and save it as '**word2vec.txt**', which is used in our assignment.

4. **GloVe**: In this approach, we first obtain the word vectors for each word in the review. Then these vectors are averaged to get the vector for the review as a whole. Two approaches were tried :

   (a) Simple averaging

   (b) Weighted averaging, with the tf-idf of the terms being the weights

   For obtaining the word vectors for each word,we use pre-trained glove word embeddings which are trained from wikipedia corpus. We extract the word embeddings of the vocabulary words of our corpus and save it as '**glove.txt**'. This file is later used in the assignment to extract the word embeddings of the vocabulary words.

5. **Doc2Vec** : In this approach, we directly create vector representations for each review using the Paragraph Vector algorithm discussed in class.We use the gensim library to facilitate the training process on our corpus, and store our trained document vector in **doc2vec.model**. When, they are to be used in GetVec.py, we simply load theis model and use the vectors.

The word2vec.txt and glove.txt are too big to be uploaded on github.While glove.txt can be downloaded from `https://nlp.stanford.edu/projects/glove/`, the word2vec.txt is formed from the google news corpus and is i the same format as the glove file.

4 Binary Classifiers:

1. **Naive Bayes Classifier from sklearn**

2. **Logistic Regression Classifier from sklearn**

3. **SVM:** LinearSVC Classifier form sklearn

4. **Neural Networks** : MLPClassifier form sklearn. A neural net of 2 layers, each with 100 neurons was created. The activation function used : ReLu. Because training the neural network takes a long time, I have saved the neural network weights after training using the pickle module in python.

5 Results

Document Representation	**Accuracies(%) for different Classification Algorithms**			
	Naive Bayes	**Logistic Regression**	**SVM**	**Neural Net**
Bag of Words	81.896	86.86	84.92	85.232
Tf-Idf	85.04	82.868	86.84	85.312
Word2Vec	77.352	85.336	85.172	82.532
Word2Vec (with Tf-Idf averaging)	51.052	82.484	78.744	79.876
Glove	75.204	85.316	85.172	82.58
Glove (with Tf-Idf averaging)	52.072	82.056	73.619	80.064
Doc2Vec	86.272	88.064	88.06	84.172

6 Conclusions

1. Clearly, some combinations produced much better results than others.

2. Across all classifiers, the Doc2Vec vector representations gave best results.This can be attributed to the fact that the algorithm used in Doc2Vec captures the semantic meaning of the document much more truly than the other approaches.

3. Also, interestingly, LogisticRegression Classifier tends to give overall best results!. This is attributed to the fact that in sklearn, returns well calibrated predictions as it directly optimizes log-loss. In contrast, the other methods return biased probabilities. Eg : GaussianNaiveBayes tends to push probabilities to 0 or 1 . This is mainly because it makes the assumption that features are conditionally independent given the class, which is not the case in this dataset which contains 2 redundant features.

4. Thus, a Doc2Vec representation with LogisticRegression Classifier gave the best results, with an accuracy of over 88%