MOVIE REVIEWS

(Project in Machine Learning CS771)

Department of CSE, IIT KANPUR

Report with Code

Submitted to:
Prof. Harish Karnick

By :
Venu Gopal Reddy (14111043)
Sushil Kumar Verma(14111038)
Rishabh Dev Shukla (14111029)
Banothu Raj Kumar (14111007)

Acknowledgement

*We would like to express our special thanks of gratitude to our instructor **Prof. Harish Karnick** who gave us the golden opportunity to do this wonderful project on the topic "**Movie Reviews**" in the course "**Machine Learning (CS771)**", which also helped us in doing a lot of Research and we came to know about so many new things. We are really thankful to him.*

Secondly we would also like to thank our group members (team) who helped us a lot in finalizing this project within the limited time frame.

Contents

About This Project

Sentiment analysis is a challenging task in machine learning. People express their feelings in language and play on words, which could be very misleading for both humans and computers.

We are given a dataset of IMDB movie reviews and expected to predict one of two sentiments good or bad based on reviews i.e. 0 (IMDB rating<5) or 1 (IMDB rating>=6).

IMDB movie reviews are the text data and we have converted those text reviews into numerical data table by using “Bag of Words”, “Tf-Idf”, and google’s “Word2Vec” techniques.

We built models such as “neural networks”, “SVM”, “LDA”, “QDA”, “Random Forest” and “Decision Tree” by varying parameters and noted the accuracies in each model in result table.

The accuracy values with “ * ” in result section of this document are the accuracies got in Kaggle submission.

About Dataset

Labeled Train Data : 25000 instances
Test Data : 25000 instances

These dataset are given on Kaggle.

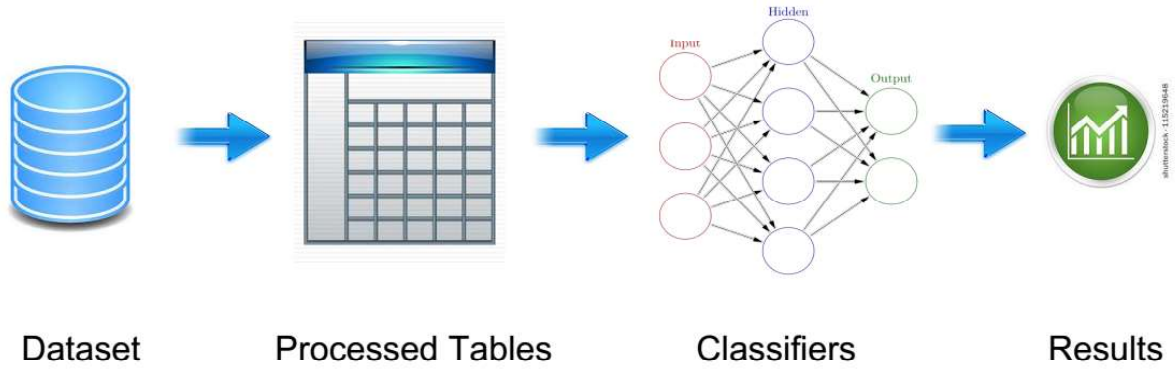
Details :

- **id -** Unique ID of each reviews
- **sentiment -** Sentiment of the review: 1 for positive reviews and 0 for negative reviews
- **review -** Text of the review

Basic Idea

Basically there are three major steps in our project:

1. processing dataset
2. creating a classifier model
3. training and testing of classifier



Bag of words

Bag of Words: It is one of the powerful technique in the natural language processing. In this model, every document is represent as a multi set of words present in the document. This model won't consider the grammar and even word ordering also. This multi set can represent as a vector form to represent the feature vector. The issue with this model is that it is not good enough to handle negation case. Negation case is that expressing thoughts using negative words like 'not', 'don't' etc. , like instead of saying 'bad movie' some one can say 'not good movie' as well.

If you use raw bag of words model, then we will loss specific nature of some words and become more general words. for example, let our vocabulary list be *[This, is, not, good, movie]* then document X, Y can be represent as below. Here each entry will represent the number of time that word occur in the document.

X : This is good movie : [1 1 0 1 1]

Y : This is not good movie : [1 1 1 1 1]

Let X is labeled as positive review and Y is labeled as negative review. If you consider the general meaning of the 'good' is positive and if any review contain this words that review tendency towards positive review. but, because of negation this word contain in both positive and negative reviews and it lost its positive nature and become a general word like 'this' that present in both the reviews.

Solution for above issue is replace k words after negation term with 'NOT_' prepended. Ideally we need to replace all words like this till punctuation symbol got appeared but most of these review writers didn't follow the punctuation symbol. In our implementation we took $k = 3$. The implementation details can be found in *Appendix a.* One more issue with this modified model that, vocabulary will become more compare to the general model.

Tf-Idf

There are still some problems with Bag of words model, that is some of the words are most frequent and that are present in each document. we can remove these words using *stop words* list. stop words are global to particular language that are most frequently used in the language. As this is global to the language, it doesn't handle the frequent words to given given data set.

One more issue that handle the rare words, these words are hardly occur in very few documents and these may not present in the any future document. These two issue can be handled by dropping these frequents words and rare words together these words are can be recognised by the Tf-Idf score of the words.

TF-IDF (Term Frequency vs. Inverted Document Frequency):
This contains the two factors Tf and Idf, by multiplying these two factors we will get the Tf-Idf score of word.

$$Tf = 0.5 + (0.5 \times f(w, d) \div max\{ f(w, d) : w \varepsilon d \})$$

$$Idf = log(N \div |\{d \varepsilon D : w \varepsilon d\}|)$$

$f(w, d)$: frequency of word in document
N : total number of documents in data sat

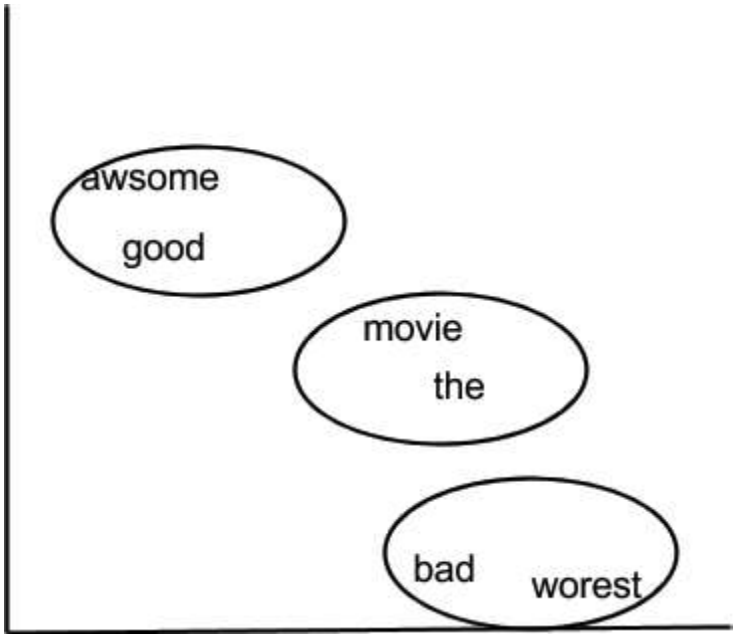
Here, Tf score will tell about the importance of word to the particular document. Idf is the scaling factor the will tell about the how frequent the word to the particular dataset. For rare and frequent words, this score is less compare to the other words. we can eliminate them by ignoring the words with less Tf-Idf scores. The representation of feature vector is just like Bag of words model, but the frequency of words is replaced with Tf-Idf scores of word.

Implementation : we use *TfidfVectorizer* from scikit learn to calculate these scores and represent feature vectors. Here we are consider the words to calculate the Tf-Idf if it is present in at least 5 documents like this we can eliminate rare words at priori.

Distributed Representation

There are some more problems we can't handle with bag of words models, as these model represents each word as a atomic quantity and it won't capture the information about similarity of words. for example, 'hotel' and 'motel' are same words by the meaning, if you represent this using bag of words model the relation between 'hotel' and 'motel' is same as 'hotel' and 'home' as this model treat all words same.

Distributed representation of word will solve this problem, by representing the each word by vector in vector space such that all similar words are in close distance than the dissimilar words, that is all similar type of words can be cluster together. For example, let our vocabulary contain the three type of words, positive, negative and neutral words.



If we assign some values in d-dimensional vector space such that all similar words are clustered together. all positive words are together, all negative words are together and far from positive words, and all neutral words are same distance to positive and negative words.

We use Google Word2Vec to represent vector embedding of words.

Google's Word2Vec

This tool provides an efficient implementation of the continuous bag-of-words and skip-gram architectures for computing vector representations of words. These representations can be subsequently used in many natural language processing applications and for further research.

The *word2vec* tool takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns vector representation of words.

There are two main learning algorithms in *word2vec* : continuous bag-of-words and continuous skip-gram. The switch *-cbow* allows the user to pick one of these learning algorithms. Both algorithms learn the representation of a word that is useful for prediction of other words in the sentence.

Several factors influence the quality of the word vectors:

- amount and quality of the training data
- size of the vectors
- training algorithm

Word2vec is also available in Python in gensim library given by Radim Rehurek.

Command in Linux to generate word2vec by terminal:

```
./word2vec -train ~/Desktop/sentences.txt -output
~/Desktop/vec.csv -size 50 -window 5 -sample 1e-4 -negative
5 -hs 0 -binary 0 -cbow 5 -iter 5
```

Decision Tree

We have varied impurity at the node by "**Complexity Parameter**" of "**rpart**" library in R, has given the best accuracy 73.06% for CP=0.1, observed accuracy when CP=0.01 which has given accuracy 70.14% and 70.89% at CP=0.02.This is due to as the CP value decreases, the height of the tree increases beyond certain height and leads to decrease in the accuracy.

Fully grown tree, using "**oblique.tree**" library has given accuracy 70.84% and after pruning the accuracy has increased to 75.32% because in pruning we remove the sections of tree that provide little power to classify instances.This technique "**growing and pruning**" is used in order to avoid horizon effect.

Splits based on number of instances are observed by varying "**minsplit**" parameter in R, has given best accuracy of 74.32% at minsplit=83, and given accuracy 72.87% at minsplit=75.As the number of instances at a node decreases(relative to optimum) the height of the increases and thus decreases the accuracy.

Random Forest

We generated Random Forests on our pre-processed data using the randomForest library of R, and did **5-fold cross validation**. First we varied the **ntree (Number of trees to grow)** parameter of our random Forest model. We started with ntree =2 , and varied it for different values till 400, our results are as follows:

ntree -	2	52	102	152	202	252	302	352	400
Accuracy-	72.8	81.288	81.74	81.7	81.81	81.92	82.064	82.008	82.01

Then we fixed ntree to be 302, for which we got the best result, and varied mtry parameter of our random forest model.

mtry : Number of variables randomly sampled as candidates at each split.
We generated random forest for ntree = 302 and mtry = (2,4,8)
The results are as follows:

mtry-	2	4	8
Accuracy-	81.62	82.148	81.936

So the best accuracy which we got from generating random forests was for ntree = 302 and mtry = 4 which was 82.148%.

Neural Networks

Neural Networks is one of the powerful technique for supervised learning. we use Bag of Model with Tf-Idf score feature representation to represent each documents. We use one hidden layer

After, construction of feature vectors of every document can be represented by the ~2,50,000 dimensional vector. All these vectors are together as a sparse matrix call data matrix. The data matrix represent this is review have lower rank because of sparsity, so it better to use dimensionality reduction to get better performance.

We use *TruncatedSVD* module in scikit learn for dimensionality reduction. This is the only active dimensionality reduction module that will take sparse matrix as input, and it is a randomization process, by using this we reduced data matrix into 300 dimensional vector.

We use simple neural network with one hidden layer are 25 neurons in it to train the data. This module is taken from '*http://danielfrg.com/*' as in scikit learn supervise neural network is not implemented yet.

By doing this we achieved our best accuracy *0.86596*, actually we can do better than by increasing number of neurons in the hidden layer but it is taking so much time to train the model.

Other Classifiers

The other classifiers which we are focusing are Naive Bayesian classifier, Support Vector Machine (SVM), LDA and QDA.

- Naive Bayesian classifier:

We are using library 'e1071' in R for bayesian classifier. With some tuning of parameters like Laplace smoothing and threshold value, we are getting an accuracy of 73.4% with five fold cross validation on training data.

- Support Vector Machine (SVM):

We are using library 'e1071' in R for SVM. With some tuning of parameters, we are getting an accuracy of 84.03% with five fold cross validation on training data.

- Quadratic Discriminant Analysis (QDA):

We are using library 'e1071' in R for QDA. With the use of straightforward function , we are getting an accuracy of 83.0% with five fold cross validation on training data.

- Linear Discriminant Analysis (LDA):

We are using library 'e1071' in R for LDA. With the use of straightforward function , we are getting an accuracy of 83.01% (slightly better than QDA) with five fold cross validation on training data.

Results

The results(accuracies) of various classifiers using various processing methods are given in the table below. Processing methods are given along rows. These are: Word2Vec, Tf-Idf and Bag of Words. The classifiers are given along columns. These are : Decision Tree, Random Forest, Neural Networks, SVM, LDA, QDA, Naive Bayesian classifier.

The values with a star mark are the accuracy which we are getting from Kaggle by submitting our classifier on Kaggle. The best accuracy , we are getting is 86.6%. Other values are accuracy of five fold cross validation result on training data.

%	Decision Tree	Random Forest	Neural Nets	SVM	LDA	QDA	Bayesian Classifier
Word2Vec	72.14	82.14	84.14	84.03	83.01	83.0	73.4
TFIDF	-	-	86.6*	80.5	-	-	-
Bag of Words	71.25	85.14*	-	-	-	-	-

Issues in Implementation

Here we are mentioning some issues, to which we are handling during our project work. These are:

1. Negation words:

   We are handling this issue in Bag of words model. We are adding negative kind of words in our vocabulary by attaching 'not' to every word after which it is taking place.

2. Useless words but not in stopwords:

   There are some words like "movie" which are useless for this dataset but are not in stopwords of english. We are treating these kind of words with Tf-Idf.

3. Large number of inputs:

   In our dataset, input size is large. It can raise a memory error or computation error with some classifier. We are sampling our dataset and choosing some less number of rows randomly to handle these errors.

4. Different types of processings on input:

   For our project, there are different kinds of processings available. These are bag of words, word to vector and Tf-Idf score. We have done every processing on input with some classifier.

5. Large corpus, tables and matrices:

   We are using dimensionality reduction where it is required to reduce size of table or matrix without affecting result.

6. Same classifier in different libraries:

   In R and Python, there are many classifiers which are developed in more than one libraries. For example : KNN, Neural nets. We have studied and used both of these libraries in project and showed results which one is better.

7. Better CPU and memory requirements:

Our project requires faster machines with enough memory. We are exchanging our laptops during implementation when it is possible and required.

Further Improvements

We can improve the accuracy of our classifiers by using some other processing techniques on input. Some techniques are:

- Term Document Matrix:

The termDocumentMatrix in R is a useful tool to make a corpus of dataset. TDM is available in library 'tm' in R. It shows which term(word) is occurring how many times in which document(review). The rows of this matrix are terms and columns are documents.

- Deep learning:

Deep learning is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using model architectures, with complex structures or otherwise, composed of multiple non-linear transformations. There are some libraries in R and Python for deep learning.

- Text Mining:

Text mining usually involves the process of structuring the input text (usually parsing, along with the addition of some derived linguistic features and the removal of others, and subsequent insertion into a database), deriving patterns within the structured data, and finally evaluation and interpretation of the output.

- Word to Phrases:

Google is developing some tool like Word2Vec which can show relations of words to phrases of documents.

RTextTools

RTextTools is a very good and new library in R for text analysis. It is available with R 3.2.0 . Some commands of this library are:
To create a document term matrix/ corpus :

```
DocTermMatrix<-create_matrix(train,language="english",
removeNumbers=TRUE, weighting=tm::weightTfIdf)
```

Make a container giving label and train/test size :

```
container<-create_container(DocTermMatrix,train$sentiment,
trainSize=1:100, testSize=101:125)
```

Create a model giving algorithm , you want to use:

```
models <-train_models(container,algorithms=c("SVM","NNET"))
```

Train your model on container:

```
results <- classify_models(container, models)
```

Watch results and other analysis of classifier:

```
analytics <- create_analytics(container, results)
```

References

Thanks to these references from where we have worked on this project :

- https://www.kaggle.com/c/word2vec-nlp-tutorial
- http://cran.r-project.org/
- http://scikit-learn.org/stable/
- http://stackoverflow.com/questions/tagged/machine-learning
- https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
- http://www.rtexttools.com/
- http://en.wikipedia.org/wiki/Machine_learning
- https://store.continuum.io/cshop/anaconda/
- https://code.google.com/p/word2vec/
- ftp://hk.cse.iitk.ac.in/
- http://danielfrg.com/