
# Project Report

Sentiment Analysis of Movie Reviews using Naïve Bayes

School of Computer Science & Technology

## Team Members:

HAQ IJAZ UL-2820150066

ZEESHAN AHMED NIZAMANI – 2820150155

DAVID MEHTTEWS CHEN - 2820150157

**Data Mining**

## Table of Contents

1. Introduction
2. Input Data
2.1 Data Source
2.2 Feature Extraction
3. Classifier Used
4. Step by step Implementation
5. References

# Introduction:

Sentiment Analysis is defined as the process to determine the attitude of a speaker or a writer With respect to some topic. Recently, it has become an active research topic, mostly due to its potential use and importance in a wide spectrum of applications of popularity analysis to product user satisfaction analysis. With the rapid growth of online information, a huge amount of data is available for the sentiment analysis. However, the sheer volume of information is a challenge itself. To separate relevant information from the irrelevant, and to gain knowledge from this unprecedented volume of huge data, an automatic analysis method is essential.

In this project we did the sentimental analysis of movies review data, using Naïve Bayes algorithm, classifier in RStudio the check and predict the effectiveness of Naïve Bayes and predict the sentiment of Test data of movies.

## 2. Input Data

## 2.1 Data Source

We get in input data for sentimental analysis from movie reviews IMDB, so we have a text file for every positive and negative reviews, in the total dataset we have 50,000 reviews, which are equally divided into 25000, train and 25000 test reviews,

So using the path of downloaded folder we can get the information from all those text files in R studio, the ratings of the reviews are set for negative reviews the score is <=4 and for positive review score is set to >=7 out of 10. So we can use these information to create a vector document names.

## 3. Feature Extraction:

In our approach as we are using the R software, we did not use the features file given along with the database, what we did id using some built in functions to extract the score of all the text files from their file names.

## 4. Classifier used:

In machine learning, **naive Bayes classifiers** are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

## 5. Step by step explanation:

In this project we used R Studio for the sentiment Analysis of given movie reviews data. R is a new and easy to learn software which is libraries enriched so easy to achieve the goal using this.

The first step we did was loading the data and information to R and perform some preprocessing, so for doing these we install some important libraries.

So using these libraries and their functions we extract the names or we can say the Meta data about all the documents and created a vector of document names.

```
> # vectors of filenames are created
> ids = sapply(1:length(all_nb), function(x) meta(all_nb[[x]], "id"))

> head(ids)
[1] "0_9.txt"     "1_7.txt"     "10_9.txt"    "100_7.txt"   "1000_8.txt"  "10000_8.txt"

> # Scores are extracted using sub funtion from file names
> scores = as.numeric(sapply(ids, function(x) sub("[0-9]+_([0-9]+)\\.txt", "\\1", x)))

> scores = factor(ifelse(scores>=7,"positive","negative"))

> summary(scores)
negative positive
   12500    12500
```

We used binary features for our model which describe the absence or presence of specific words in the dictionary. So we should expect that words like (boring, bad, and horrible) as negative and words like (inspiring, enjoyable, etc) as positive.

After these steps a document term matrix is created which in this case have **118235** columns which shows that we have found this number of different words in the data, the matrix we get is very sparse means most of the entries are 0, after examining the matrix we checked it how sparse it is.

```
> # document term matrix (dtm) is created
> dtm_nb = DocumentTermMatrix(all_nb)

> dim(dtm_nb)
[1]  25000 118235

> dtm_nb
<<DocumentTermMatrix (documents: 25000, terms: 118235)>>
Non-/sparse entries: 2493738/2953381262
Sparsity           : 100%
Maximal term length: 64
Weighting          : term frequency (tf)
```

Then infrequent items were removed and we tried to repeat this with different sparse

```
> # infrrequent items are removed
> # trying to repeat this with different sparsity
> dtm_nb = removeSparseTerms(x=dtm_nb, sparse = 0.99)

> dim(dtm_nb)
[1] 25000  1603

> dtm_nb
<<DocumentTermMatrix (documents: 25000, terms: 1603)>>
Non-/sparse entries: 1532336/38542664
Sparsity           : 96%
Maximal term length: 14
Weighting          : term frequency (tf)
```

All these techniques are applied to get the most general feature data which we will pass to the Naïve Bayes Classifier for training it, for this we use the naiveBayes() function from the package named as "e1071", so what we do is to provide the naïveBayes() our feature data frame and another argument is the vector of the output labels.

```
> library(e1071)

> model_nb = naiveBayes(train_df_nb, train_scores)
```

For obtaining the predictions of our training data we use the function Predict ()

```
predictions_train_nb = predict(model_nb, train_df_nb)
save(predictions_train_nb, file = "predictions_train_nb.RData")

> mean(predictions_train_nb == train_scores)
[1] 0.82525

> table(actual = train_scores, predictions = predictions_train_nb)
          predictions
actual     negative positive
  negative     8454     1553
  positive     1942     8051
```

From the above diagram, we can see that we have hit round about 83% of accuracy with our Naïve Bayes Model. Now we will check it for our test data as well.

```
predictions_test_nb = predict(model_nb, test_df_nb)
save(predictions_test_nb, file = "predictions_test_nb.RData")

> mean(predictions_test_nb == test_scores)
[1] 0.8202

> table(actual = test_scores, predictions = predictions_test_nb)
          predictions
actual     negative positive
  negative     2088      405
  positive      494     2013
```

From the figure above we can see that the result of test data is also about 82 percent which we can compare with the train data result, still there are number of things which need to be improved, e.g the word movie and movies are same but inflected, so these kind of problems we will try to solve applying a preprocessing step called stemming, so we again train the naïve Bayes and see the result below

```
> # a preprocessing step stemming is performed
> all_nb = tm_map(all_nb, stemDocument, language = "english")

> dtm_nb = DocumentTermMatrix(all_nb)

> dtm_nb = removeSparseTerms(x=dtm_nb, sparse = 0.99)

> dtm_nb = weightBin(dtm_nb)

> df_nb = as.data.frame(as.matrix(dtm_nb))

> train_df_nb = df_nb[sampling_vector_nb,]

> test_df_nb = df_nb[-sampling_vector_nb,]

> # train the model on the corpora of stemmed
> model_nb_stem = naiveBayes(train_df_nb, train_scores)

> if (file.exists("predictions_test_nb_stem.RData")) {
+   load("predictions_test_nb_stem.RData")
+ } else {
+   predictions_test_nb_stem = predict(mo .... [TRUNCATED]

> mean(predictions_test_nb_stem == test_scores)
[1] 0.804

> table(actual = test_scores, predictions = predictions_test_nb_stem)
          predictions
actual     negative positive
  negative     2076      417
  positive      563     1944
```

The result of stemming process is 80% which is even lower than the previous results, so it means that stemming process not always improve the result, hence we have second possible way to improve the results is additive smoothing process which is also called "**Laplacian method".**
What we do is , using our original document matrix, we ad additive smoothing to our naïve bayes model by applying some laplace parameter. But for the dataset we using , we still don't witness any improvement by this method.
See the code and results in the picture below.

```
> model_nb_laplace = naiveBayes(train_df_nb, train_scores, laplace=10)

> if (file.exists("predictions_test_laplace_nb.RData")) {
+   load("predictions_test_laplace_nb.RData")
+ } else {
+   predictions_test_laplace_nb = p .... [TRUNCATED]

> mean(predictions_test_laplace_nb == test_scores)
[1] 0.7882

> table(actual = test_scores, predictions = predictions_test_laplace_nb)
          predictions
actual     negative positive
  negative     2101      392
  positive      667     1840
>
```

The result of accuracy in this case is reduced to round about 79%...

**References:**

[1] www.imdb.com

**[2] Mastering Predictive Analytics with R (Book)**

[3] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher
Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting*
*of the Association for Computational Linguistics: Human Language Technologies - Volume 1* (HLT
'11), Vol. 1. Association for Computational Linguistics, Stroudsburg, PA, USA, 142-150.

[4] Vladimir N. Vapnik, *The Nature of Statistical Learning Theory*. Springer, 1995.

[5] Wikipedia.org