# Naive Bayes Classifier in Local and MapReduce Mode

**Karan Malhotra, M.Tech(SE), Sr No-14532**

## 1. Introduction

Naive Bayes Classifier is the probabilistic classifier using bayes theorem for the posterior calculation with a naive assumption that the features are independent of each other.The following equations describes naive bayes classifier approach.
$p(y|x) = \frac{p(x1|y)p(x2|y).....p(xn|y)p(y)}{p(x)}$
$y* = argmax_y(p(x1|y)p(x2|y).....p(xn|y)p(y))$

## 2. Dataset

The dataset provided is multi label document classification dataset and its details are as follows:
**1.** Training instances: 214998
**2.** Validation instances: 61497
**3.** Testing instances: 29997
**4.** Total Classes: 50

## 3. Naive Bayes implementation in local

This section describes the implementation of the naive bayes classifier in the local machine mode where there was no use of distributed frameworks utilized all the calculations and hashing was done in memory only for given dataset.

### 3.1. Preprocessing

**1.** All the punctuation and single length characters were removed while preprocessing.
**2.** All the words were then converted to lower case .
**3.** Porting or Stemming of the provided data sets were not performed.

### 3.2. Training procedure

**1.** A Dictionary was built which included key as a word and the value was a list with a length as number of unique class labels and every entry of this list is the word count of the word in the respective class label.($Count(W = w, Y = y)$)
**2.** A dictionary was also created to store the number of times a particular class label occurs. ($Count(Y = y)$)
**3.** The Validation is also performed to tune the value of hyper-parameter K(Add k smoothing ) using development data set provided.

### 3.3. Parameters

**Total Trainable Parameters:** The trainable parameters mainly depend on the following counts:
$Count(Y = *)$ , $Count(Y = y, W = *)$ , $Count(Y = y)$
Therefore total trainable parameters come out to be = Vocab-size*No-of-labels + No-of-labels **=4622325.**
**HyperParameters:** Only one hyper-parameter K which is required for add K smoothing.

### 3.4. Testing

**1.** The following equation describes how the testing phase is carried out.
$log(p(y|w_1, w_2, .....)) =$
$\sum_i log\frac{C(Y=y,W=w_i)+K}{C(Y=y,W=ANY)+K*|V|} + log\frac{C(Y=y)}{C(Y=ANY)}$
where K is smoothing factor and V is vocabulary.

**2.** For every given test example the $log(P(Y = y|w_1w_2...))$ is calculated for each y and argmax over y is taken and the argmax over y is assigned as predicted label to the respective example.
**3.** If the predicted example is given in the list of true labels provided in dataset corresponding to that example then the prediction is termed as correct.

### 3.5. Results

**Development Accuracy:** 79.87%
**Testing Accuracy :** 81.08%
**Train time :** $704secs$
**Test time :** $159secs$

## 4. Naive Bayes Implementation in Hadoop framework using MapReduce

The data pre-processing step is same as the one performed initially in memory based naive bayes. Overall 5 map-Reduce jobs were performed to train and test the model for the given datasets.

### 4.1. Training procedure

Only one Mapper and Reducer was used for training purpose as the training phase in this case was just to spit out the counts of all the word label combinations.
**Mapper:** The mapper was reading every document(a single document was a line in data set) doing pre-processing separating labels and data and then spitting following values:
For each label in given labels-
$(!total, 1)$ For calculating Count(Y=ANY)
$(label\%\%, 1)$ For calculating Count(Y=label)
For each word in doc:
$(label\%\%\%, 1)$ For calculating Count(Y=label,W=ANY)
$(label\Lambda word, 1)$ For calculating Count(Y=label,W=w)
$(word, 1)$ For calculating vocabsize.

**Reducer:** This performs the function same as of a basic reducer which just aggregate just the counts of a particular key. The output of the reducer is like:
$(label\Lambda word, count)$ Eg: (British_filmsΛ dreams,13)

**Training Time:** The Best training time was observed for when 10 Reducers were used in parallel and it was **65**secs. The below Figures shows the variation in time with increasing number of reducers

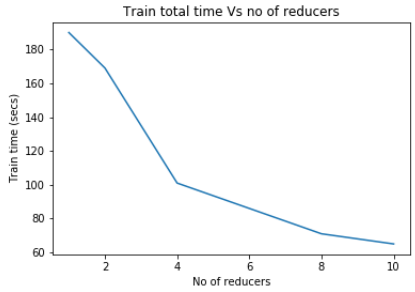


*Figure 1.* Total Train Time vs No of reducers

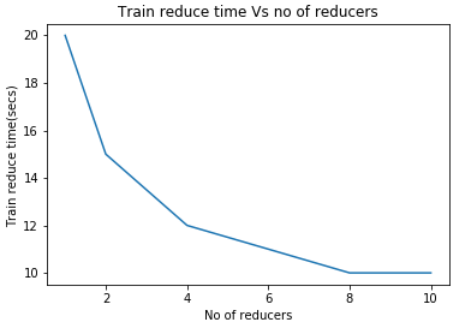


*Figure 2.* Train Reduce Time vs No of reducers

### 4.2. Testing procedure

Overall 4 Map-Reduce jobs were performed to do testing of the given data-sets. To reduce the mapper and reducer code complexities more number of mapreduce jobs were preferred rather than trying to do in one or two mapreduce jobs.
**Mapper1:** The input of this mapper is the testfile and the mapper reads every line of the test file and prints the key values whose count values are required for testing so that in further stages only those counts are extracted from training counts which are necessary for this test file.
The necessary keys whose counts are required would be the (Y=ANY), (Y=label), (Y=ANY,W=w) and (Y=label,W=w) for all w which are the words present in test file and all 50 labels.
**Reducer1:** This reducer will take the output of the training MapReduce job as cacheFile and for each unique key spitted by mapper will read the cacheFile and spit the training count of the particular key.So this MapReduce job has spitted a test dictionary kind of thing which contains all the key counts required for testing.

**Mapper2:** The input of this mapper is the test file with line numbers inserted in each line of the file.The mapper job is to spit the line number and word as the key so that in further reduction strategy the word probability is calculated with line number also preserved.The value for every key is given is 1 as when it is further aggregated will give the word count in particular line number.
For Eg:$((4, dream), 1)$ 4 denotes the line number and dream is the word.

**Reducer2:**The reducer will take the test dictionary produced by Mapper1 and reducer1 as cacheFile and for each key it will calculate the probability of the word with every label (P(Y=label,W=w)) and print the key value pair as $((egid\Lambda label\Lambda word), score)$ where score is log probability multiplied by the count of the word in that particular line(spitted by mapper) with smoothening incorporated.For each label the prior probability score is also spitted out for further calculations.
**Mapper3 and Reducer3**: This MapReduce job is just to aggregate all the scores for the particular egid(line number) and label with all words score for particular these two entries are summed.So the output of this mapReduce job is like :
$(10000\Lambda Black\ and\ white\ films, -508.08)$ where 10000 is the line number and Black-and-white is the label and the value is the log of $P(W = w1, w2.....wn|Y = y)P(Y = y)$.
**Mapper4 and Reducer4**: This will take the input as the output form mapper3 and reducer3 framework and then it will perform a max func on all the labels for particular

egid and to print the label for a particular eg-id whose score value is maximum.It will print output like:
$(10000, American\_male\_film\_actors)$ therefore the predicted value for line number 10000 is the label American_male_film_actors.
To achieve this, partition of mapper output for reducers is done on basis of egid. And then reducer performs the max over all the labels for a particular eg id.
At the end the output of the last mapReduce job and the original testfile are given to a python script file and it prints out the accuracy achieved by the naive Bayes classifier.

The below Figures shows the variation in time with increasing number of reducers

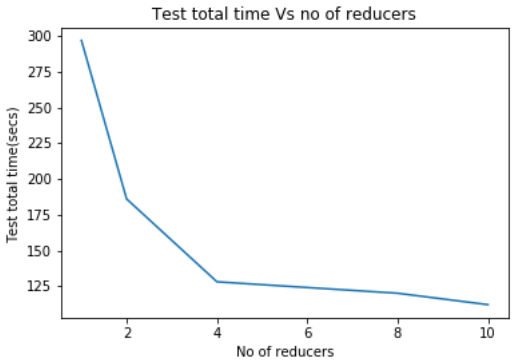


*Figure 3.* Total Test Time vs No of reducers

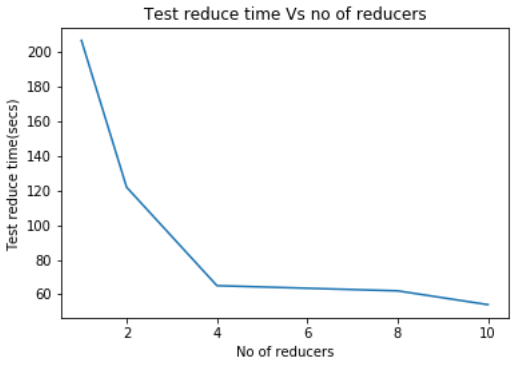


*Figure 4.* Test Reduce Time vs No of reducers

**Testing Time:** The Best testing time was observed for when 10 Reducers were used in parallel and it was **112**secs.
**Development Accuracy:** 78.87%
**Testing Accuracy:** 80.04%

## 5. Conclusions

**1.** It was clearly observed that when the number of reducers were increased the training and testing time both decreased but they can't be termed as linear as initially the rate of decrease in time is very high and afterwards its somewhat decreases with very less margin.
**2.** This can be explained as the number of mappers and reducers are continuously increased the shuffle and sort times are also added to the overall time therefore after a certain number of reducers the job time is not improved by a big margin.
**3.** It was also observed that the training time was reduced to **65 secs** from **704 secs** and the overall training and test time was decreased to **177 secs** from **863 secs** when ten reducers were used.Therefore a nearly 12 time improvement is seen in training time and 5 times improvement in total time when distributed mapReduce framework is used.