# Document Classification using Naive Bayes Classifier

**Sahil Deep** [*1]

## Abstract

In this assignment, the training and testing of a Multinomial Naive Bayes Classifier for classification of DBPedia dataset was implemented with in-memory and streaming based Map Reduce approach.

## 1. Preprocessing on Dataset

- Each document was converted into words using $wrds = re.findall('w+', document)$ as tokenizer.
- Words were converted to lower case.
- No stopword removal or stemming was done.
- As a result, the vocabulary consisted of 301404 words(for full training data).

## 2. Basic Methodology

- On the given training and test dataset initialized an event counter(hash table) $C$.
- For each example $id, y, x_1, ....x_n$ in training dataset: $C = (\text{“}Y = ANY\text{”}) + +; C(\text{“}Y = y\text{”}) + +$
- for $j$ in $1, .....d_{id}$ :
  $C(\text{“}Y = y\text{”} \wedge x = ANY) + +; C(\text{“}Y = y \wedge X_j = x_j\text{”}) + +;$
- For each example $id, y, x_1, ....., x_d$ in testing dataset
  (a) For each $y'$ in domain($Y$)
  (b) Compute $Pr(y', x_1, .....x_d)$ as $a + b$ where $a$ and $b$ are as follows:

$$a = \frac{\sum_{j=1}^{d} log(C(X = x_j \wedge Y = y_j) + mq_j}{C(X = ANY \wedge y = y') + m} \quad (1)$$

$$b = \frac{logC(Y = y') + mq_y}{C(Y = ANY) + m} \quad (2)$$

- Return best $y'$

## 3. In Memory Implementation

In Local Naive Bayes' implementation the algorithm is implemented same as mentioned in Section 2 with in memory hash table.

## 4. Parameters

Total number of parameters are 1195617. Out of these 1195517 are of type $C(X = x_j \cap Y = y')$, 50 are for $C(X = ANY \cap Y = y')$ and 50 are for the class frequency.

## 5. Hadoop Implementation

- Reorganize $C$ to make $C'$.
  - Stream through $C$ and for each event count $(e, n_e)$, let $W$ by the word such that $e = \text{“}Y = y$ and $W = w\text{”}$. Print the message $(w, e, n_e)$ to temporary file 1.
  - Sort temporary file 1 by word $w$.
  - Stream through the sorted temporary file 1 and convert it to pairs $(w, wordCounts_w)$, using the fact that all the event counts for $w$ are now stored sequentially. Store the output in file $C'$.
- Produce word-count requests: For each document $i, (w_1, ..., w_{ni})$ :
  - For each word position $j$ in document $i$, write a message $(w_j, i)$ to temporary file 2.
- Produce word-count replies:
  - Concatenate $C'$ and temporary file 2, sorting them by word, so that the word-count records from C' precedes the request from temporary file 2.
  - Stream through the resulting file as follows:
    - If a line contains a tuple $(w, wordCounts_w)$, save that record in memory.
    - If a line contains a tuple $(w, i)$, then print a message $(i, w, wordcounts_w)$ to temporary file 3.
  - Sort temporary file 3 by document id $i$.
  - Stream through temporary file 3, accumulating together all the sequentially contiguous tuples

*Table 1.* Training Time

| LOCAL/REDUCERS | TRAINING TIME |
|---|---|
| LOCAL | 0M25.68S |
| MAPREDUCE(R=1) | 4M7.39S |
| MAPREDUCE(R=2) | 3M34.63S |
| MAPREDUCE(R=5) | 3M17.98S |
| MAPREDUCE(R=8) | 3M4.69S |
| MAPREDUCE(R=10) | 3M1.23S |

*Table 2.* Testing Time

| TESTING TIME | FULL TEST | FULL TRAIN |
|---|---|---|
| LOCAL | 0M51.09S | 10M34.34S |
| MAPREDUCE(R=1) | 3M19.29S | 16M33.30S |
| MAPREDUCE(R=2) | 3M12.42S | 22M48.57S |
| MAPREDUCE(R=5) | 3M22.053S | 15M42.29S |
| MAPREDUCE(R=8) | 3M19.25S | 16M40.74S |
| MAPREDUCE(R=10) | 3M34.09 | 15M33.32S |

$(i, w, wordCounts_w)$ for a single document $i$, and saving them in memory. When this sequentially contiguous block is finished, output a message $(i, \hat{y})$ where $\hat{y}$ is the predicted class for $i$.

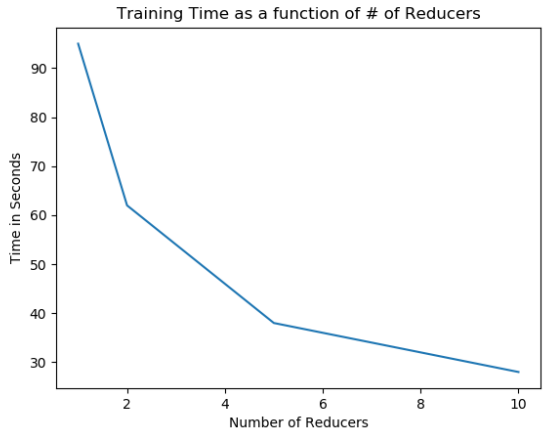


*Figure 1.*

## 6. Experimental Analysis

The figure 1 above plots the training time with respect to number of reducers. The relation is not linear and we can observe that increasing the number of reducers decreases the training time to some extent but later the the decrease in training time is not that prominent due to increased communication overhead.

*Table 3.* Classification Accuracy

| IMPLEMENTATION | TRAINING | DEVELOP | TESTING |
|---|---|---|---|
| LOCAL | 95.62% | 75.87% | 77.89% |
| MAPREDUCE | 95.62% | 75.87% | 77.89% |

*Table 4.* Reducer vs Time

| REDUCER COUNT | TIME(S) |
|---|---|
| 1 | 95 |
| 2 | 62 |
| 5 | 38 |
| 8 | 32 |
| 10 | 28 |

Since the algorithm implemented in both the cases is the same thus both of them have the same accuracy over training,development and test datasets.

The Reducer vs Time in the table 4 is measure for stage 1 of the training by measuring the time between mapper $100\%$ reducer $0\%$ to mapper $100\%$ reducer $100\%$ on training dataset. This measurements been used for plotting the figure 1.

## References

Along with lecture slides the following pdf http://www.cs.cmu.edu/~wcohen/10-405/prob-tour+bayes.pdf was used to implement in memory Naive Bayes'.
For Hadoop implementation used the following pdf http://www.cs.cmu.edu/~wcohen/10-605/notes/scalable-nb-notes.pdf. The steps mentioned above are taken from this pdf.