Naive Bayes on Hadoop Assignment 1

Udit Gupta (14454) [1]

1. Methodology

The steps followed in predicting the class of a document are as follows:

1. Count the Number of occurrences of each word in the training set. The mapper will emit (K,V) pairs where 'Key' will be the word and 'Value' will be the class to which it belongs. Reducer will aggregate words from same class together. The output of this step is:

   (word: (class, #occurences in class))

2. Count the Number of occurrences of each class and the number of words in each class. This outputs:

   (class: (#occurences Of Class, #words))

3. Before the classification step we need to build a class count dictionary

4. The class count dictionary is built keeping in mind the test set requirement. Output of this step is:

   (line_no, test_class: (word:(train_class,#occurences in class)))

2. Algorithm

The Formula for Naive Bayes is straight forward and is implemented using log probabilities to avoid underflow. However, plain Naive Bayes won't work as some words may have never occurred for a certain class and hence it will assign zero probability to it and hence total probability will be zero.
To overcome this limitation smoothing is used to hallucinate a few extra counts for each word. The denominator is then normalized such that the total probability sums to 1.
I wrote two algorithms which differ slightly in terms of the expression of smoothing applied to Naive Bayes.

2.1. Implementation 1

$$logPr(x_1, ..., x_d, y^*) = log\frac{C(Y = y^*) + mK}{C(Y = ANY)} + \sum_i log\frac{C(X = x_i, Y = y^*)}{C(X = ANY, Y = y*) + \frac{m}{K}}$$

2.2. Implementation 2

$$logPr(x_1, ..., x_d, y^*) = log\frac{C(Y = y^*) + mK}{C(Y = ANY) + m} + \sum_i log\frac{C(X = x_i, Y = y^*) + mK}{C(X = ANY, Y = y*) + m}$$

K = $(\#Unique\ words\ in\ training\ set)^{-1}$
m = 1 (laplace smoothing)

3. Performance on DBPedia dataset

3.1. Implementation 1

Accuracy

Architecture	Train	Validation	Test
Local	68.5	66.7	65.4
Cluster	54.2	50.3	53.0

Time (in mins)

Architecture	Train	Validation	Test
Local	175	58	30
Cluster	45	14	8

3.2. Implementation 2

Accuracy

Architecture	Train	Validation	Test
Local	47	32.3	28.6
Cluster	69.1	61.8	65.0

Time (in mins)

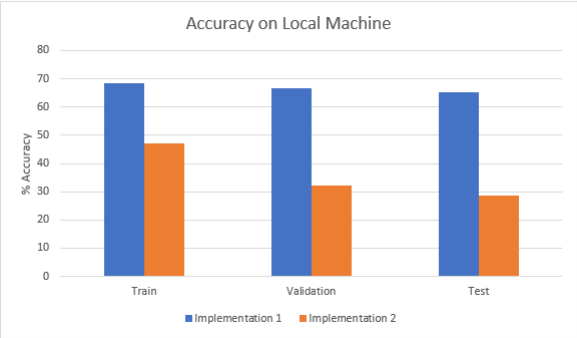
Architecture	Train	Validation	Test
Local	170	56	28
Cluster	44	14	8

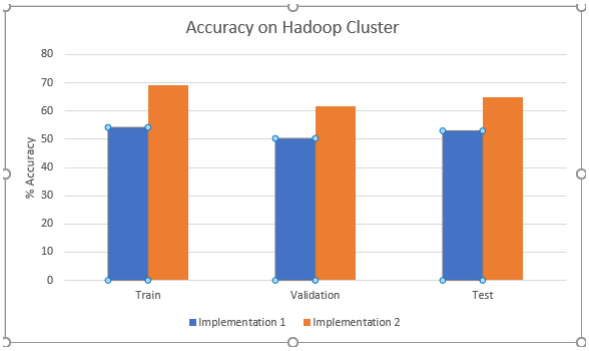
Note: *Number of Reducer is set to 1*

[1]Department of Computational and Data Sciences, Indian Institute of Science.

Assignment1 for the course Machine Learning with large datasets (DS-222)

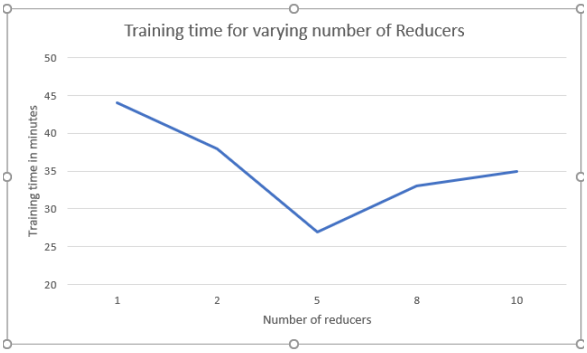
4. Inference





Implementation 1 performs better on the local machine, whereas *Implementation 2* performs better in a cluster setting.

5. Varying number of Reducers



Observations: The wall time is non-linear with respect to the number of reducers. Initially the wall time decreases as the number of reducers increase. In this case the optimal number of reducers to use is 5. Beyond this, the wall time increases. This is because instantiating a reducer has an overhead cost attached to it and also, the data needs to be split across more number of reducers which results in network bottleneck. Wall time comprises of both, the overhead time and the computation time. As the number of reducers increase, the increase in overhead time is far more than the decrease in computation time and hence the wall time increases beyond a certain level.