Machine Learning with Large Datasets- Assignment 1
Implementing Naive Bayes In Memory and using Map Reduce

Shubham Bansal(14666)

1. Probem Statement

Given is a multi-class multi-label classification problem and we need to implement naive bayes classifier comparing the computational performances when implemented in memory and when implemented with mapreduce using multiple mappers and reducers.

2. Data Preprocessing

Following are the preprocessing done on training,development and test data available:

- Cleaning of html links
- Removal of punctuation marks
- Removing alphanumeric/numeric words
- Converting to lowercase
- Removing words with length less than or equal to 2

3. In Memory Naive Bayes

Using 16GB RAM laptop with no GPU and no parallelization support.

3.1. TRAINING

Since, We have 50 classes in training data, we have created 50 dictionaries containing word counts corresponding to each of the labels. Each of these dictionaries has word as key and counts as value. Also, we have created two more dictionary one containing the prior probability and another containing the count of total number of words corresponding to each label.

Total vocabulary size: 253227
Total trainable parameters: 12661450
Training time: 272 seconds

3.2. TESTING

Corresponding to each test document, for each word present in it, we sum log of probabailty of it being coming from the given label. If the word is not present in training vocabulary, we ignore it. Else, we do the laplace smoothing with weight of 0.01.
Total vocabulary size: 253227
Total trainable parameters: 12661450
Testing time on test data: 92 seconds
Accuracy: 81.2 %

4. Naive Bayes using MapReduce

The MapReduce implementation has been experimented on turing cluster. More details about this cluster can be found from **turing.cds.iisc.ac.in**

4.1. TRAINING

Here we have used one mapper and one reducer file.

MAPPER 1:
This mapper reads the training data, preprocess it and finally emit key value pairs as below:
{**word&label, 1**}
Where key is word and label seperated by tab and value is 1 corresponding to each emission Following is the example:
{**across&Villages_in_Turkey, 1**}
Following is the example:
{**across&Villages_in_Turkey, 1**}
Also, we emit corresponding to each word for a particular label as below:
{**ANY&label, 1**}
Following is the example:
{**ANY&Villages_in_Turkey, 1**}
This will help us to keep count of number of words present in the training vocabulary corresponding to each label which will be required to calculate the probabilities.
Also, we emit corresponding to each training example:
{**P_ANY&label, 1**}
Following is the example:
{**P_ANY&Villages_in_Turkey, 1**}
This will help us to find the prior probabilities.

REDUCER 1: The reducer finally aggregates the counts of key,value pairs emiited from mapper. It will emit following key,value pairs of the form below:

{**word&label, count**}
{**ANY&label, count**}
{**P_ANY&label, count**}
where count is aggregated sum of values.

Total vocabulary size: 253227
Total trainable parameters: 12661450
Training time: Best overall mapreduce time obtained using **10 reducers** of **120 seconds**. Also, corresponding wall clock reduce time is **4 seconds**. Testing time with number of reducers can be observed from the following plots:

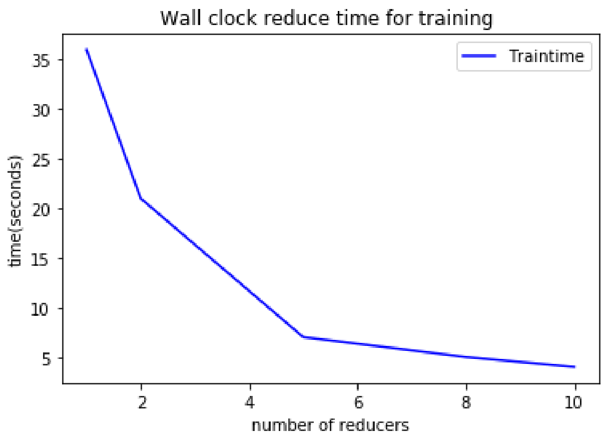

Figure 1. Total Reducer Wall Clock time vs No of Reducers

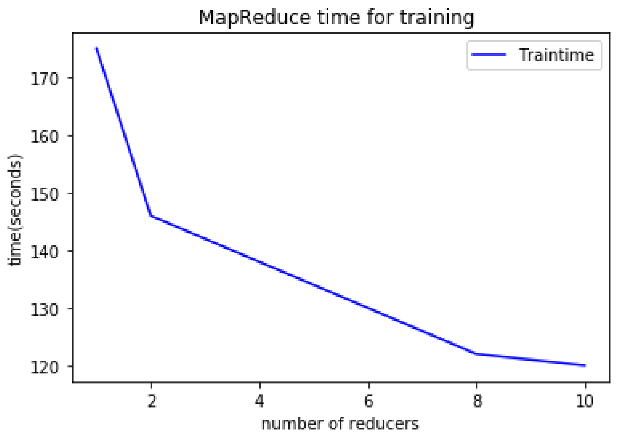

Figure 2. Total MapReduce time vs No of Reducers

4.2. CACHEFILE

The output file of reducer1 will be used as cache file for testing part as we need to read the trained parameters to make the predictions for the test data.

4.3. TESTING

This comprises of 3 mapper file and 3 reducer files. We describe function of each mapper and reducer below:

MAPPER 2:
This mapper reads the training data, preprocess it and finally emit key value pairs as below:
{**word&index, 1**}
Where key is word and index number seperated by tab and value is 1 corresponding to each emission. Following is the example:
{**across&8488, 1**}
Note, index number is obtained by incrementing it with each example while iterating test data.

REDUCER 2:
The reducer finally aggregates the counts of key,value pairs emiited from mapper. It will emit following key,value pairs of the form below:
{**word&index, count**}
where count is aggregated sum of values.
Following is the example:
{**across&8488, 6**}

MERGING
Now, we merge the output files of reducer1 and reducer2. Our aim is to bring the index count and label count of each word together and further use word as key. Following is the key value pair form, we will be using for input to next mapper:
{**word, index&count**}
{**word, label&count**}

MAPPER 3:
This mapper is an identity mapper which again emits the following key,value pair:
{**word, index&count**}
{**word, label&count**}

REDUCER 3:
Now, since word is the key, hence, we have all information related to any particular word on a single reducer. Also, we obtain the trained parameter through cache file. Hence, correponding to each word , it adds the log of probability of that word for given label and index. Hence, We have saved the score for each index and label pair in a dictionary which we emits as below:
{**index&label, score**}
Following is the example:

{**0, American_comedy_films&-485.18**}
{**0, American_drama_films&-434.80**}
{**0, American_film_actresses&-409.18**}
{**0, American_film_directors&-445.18**}

Advantage of doing it this way is that we need only prior information from cache file and hence, entire cache file has not to be read into a dictionary and hence, makes it very scalable.

MAPPER 4:
This mapper is an identity mapper which again emits the following key,value pair:
{**index, label&score**}

REDUCER 4:
Following is the sorted input to reducer 4:
{**0, American_comedy_films&-485.18**}
{**0, American_drama_films&-434.80**}
{**0, American_film_actresses&-409.18**}
{**0, American_film_directors&-445.18**}

For each index, this reducer compares the score of each label which is the sum of input score added with the log of prior probability obtained from cache file. Finally, it emits the predicted label corresponding to each index. Following are the predicted results for first few examples in given test set:
{**0, Guggenheim_Fellows**}
{**1, American_drama_films**}
{**2, The_Football_League_players**}
{**0, Indian_films**}

Accuracy: 81.2 %

Testing time: Best overall mapreduce time obtained using **10 reducers** of **95 seconds**. Also, corresponding wall clock reduce time is **26 seconds**. Testing time with number of reducers can be observed from the following plots:

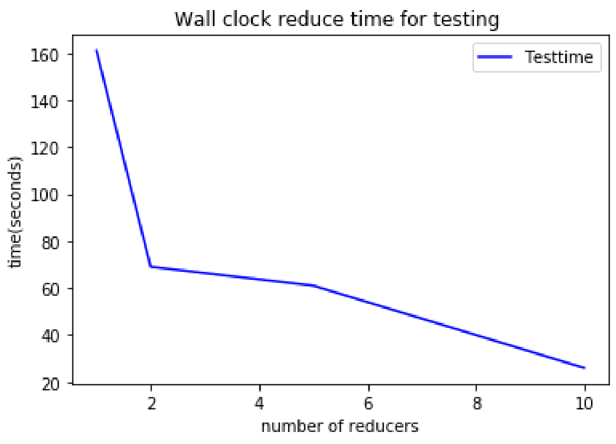

Figure 3. Total Reducer Wall Clock time vs No of Reducers

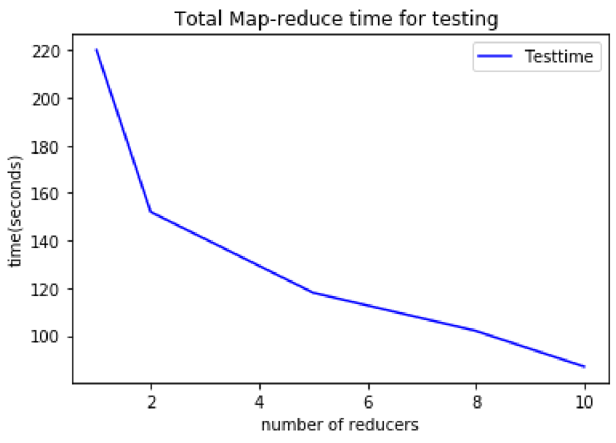

Figure 4. Total MapReduce time vs No of Reducers

4.4. Conclusion

Following are the observations from the experiment.

- We observe that for MapReduce tasks, both reducer wall clock time and overall mapreduce time decreases significantly with increase of number of reducers for both training and testing.

- In comparison to InMemory implementation, overall testing time is less than overall MapReduce time with 10 reducers. This may be because of the fact, that we have experimentated naive bayes on small dataset which can easily fit into memory. Hence, we could not see improve in performance.

- Performance may be more evident with large datasets.

- In case of training, we observe marginal improvement with MapReduce.