Naive Bayes Classifier: Local and MapReduce Implementation Comparison

Kapil Pathak [*] [1]

Abstract

The assignment mainly consists of the in-memory implementation as well as Hadoop Map-reduce implementation of Naive-Bayes Classifier. As a learning point of view, this assignment was very useful to grasp the concepts of streaming methods, hadoop map-reduce implementation in Java, working and getting familiarized with Turing cluster of SERC, IISc.

1. Implementation Details

In the current assignment, Naive-bayes classifier has been implemented in ipython notebook as a dictionary counting manner as well as streaming method. But the streaming method has only been implemented till we get a tuple consisting document id in test dataset as key and a tuple consisting a word appearing in the test document and its corresponding frequencies for class labels in training dataset as a value. To implement the streaming algorithm, has been referred. These implementations are in python.

For Hadoop implementation, the training phase is completed and the graph of time required to complete reduce task vs number of reducers can be obtained. This implementation is in Java. For testing phase, more debugging is required to come up with working classifier.

This report talks about following points about the implementations

- Methods of implementation in local as well as in Map Reduce framework.
- The effect of streaming on the counting process
- Accuracy of the model in training, development and testing dataset
- Train and Test timings of both implementations
- Effect of number of reducers in training phase

[*]Equal contribution [1]Department of Computer Science and Automation, Indian Institute of Science, Bangalore. **AUTHORERR: Missing \icmlcorrespondingauthor.**

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

1.1. Methods of implementation in local as well as in Map Reduce framework

In the assignment, an attempt is made to compare two methods in the local. One method uses regular counting which uses the dictionary of dictionary from Python. In this method, the small chunk of DBPedia dataset is directly loaded into the memory and each document is loaded at a time and processed with the removal of stop-words and lemmatization. We get dictionary of dictionary containing documents with their words and their corresponding frequencies. Once a testing documents comes, we process the documents and find corresponding word log probabilities and sum them up. Naturally this increases the seek time to locate the dictionaries as well as it uses considerable main memory. This implementation is given in ipython notebook.

In another attempt, streaming naive bayes is implemented in which we read the dataset into the buffer but just a line at a time and process it. The advantage of this method is that it uses constant amount of memory as it exploits the sequential sorted messages given from one step to another. This implementation has been performed by following the (Cohen, 2017) This part also has been discussed with Mr. Vikram Bhatt from CDS, IISc.

In the third implementation, the same classifier has been implemented in Hadoop MapReduce framework in Java. In this framework, we apply one MapReduce stage at the training phase while apply another MapReduce stage at the testing phase. Here, training phase is completed in which we generate word frequencies for each label in the training dataset. While in the testing phase, the HashMaps required to store various counts have been put into **setup** method. Other tasks such as reading a test document, log probabilities calculation and classification have been performed into **map** method. Here training phase code partially has been referred from a book, ”Hadoop Definitive Guide”, by Tom White (White, 2009). It is also discussed with Mr.Vikram Bhatt, CDS, IISc. Testing phase code has been discussed with Mr.Nilesh Agarwal from CSA IISc. Testing phase code requires more debugging.

1.2. The Effect of Streaming on the Counting Process

As streaming requires reading a line one by one, it uses a constant memory. In the first method where we load whole dataset into the memory, the seek time for dictionary slows down the counting process significantly. While in the streaming process, we send the messages to the **sort** of Linux which uses a constant memory queue and sort the messages. In the next stage, program uses constant memory to get the count of particular key value pairs. The difference between normal counting process and streaming process has been observed significantly as the first method takes 2-3 min to process each document in the testing set. Training phase of the first method took 62 seconds on the small chunk of DBPedia dataset that Prof.Partha made us available earlier while the testing phase of each document took 120-130 seconds on an average. On the other hand, training full dataset took 246 seconds through streaming phase.

1.3. Accuracy of the Model in Training, Development and Testing dataset

In the first implementation, the training accuracy has been performed on only 100 documents each from training dataset, testing dataset and development dataset as it was taking more around 1.5-2 minutes on each document in each dataset. The accuracy in training dataset is around 81 percent for first 100 documents, while it drops to 31 percent in development dataset and 23 percent in the testing dataset.

Other two implementations of streaming in local and Hadoop MapReduce have been stuck at testing phases. The training phases for both implementations are completed.

1.4. Training and Testing phases Clock Timings

In the first implementation, the training phase takes 62.82 seconds while testing phase takes 11614.93 seconds for testing 100 documents. In streaming implementation, training phase takes 24.83 seconds. The training phase in the streaming phase has been referred till the step where we generate sorted tuples where each tuple consists of an index of a testing document, word in the testing document and an array consisting of training dataset frequencies for that word in each label as given in the (Cohen, 2017).

1.5. Effect of Number of Reducers in Training Phase

A graphs has been plotted to study the effect changing number reducers in the training phase of the Hadoop MapReduce implementation. In the graph, X-axis shows the number of reducers while Y-axis show the time taken between map 100% reduce 0% and map 100% reduce 100%" from syslog for full dataset.

Here we can observe that the time taken to complete reduce

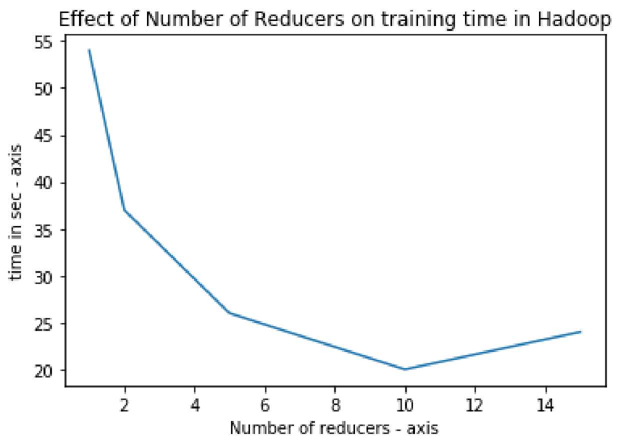

Figure 1. Variation of time taken to complete reduce task with number of reducers

step decrease as we go on increasing number of reducers till number of reducers reaches 10 and again starts raising again beyond 10. Here if we keep too few number reducers, it creates load imbalance in the task as the task gets distributed among reducer workers. But if we keep too many number of reducers, it also increases the cost of communication among the reducers. So eventually increases the time taken to complete the whole task.

Another observation is that number of splits are only 1. The reason can be the block size in HDFS is 128 MB while the size of full dataset is less that the block size. Hence this limits the number of splits to 1.

Acknowledgements

I sincerely thank to Prof. Partha for his efforts to make us understand the concepts of Steaming Naive Bayes algorithm. Also I wish to acknowledge Mr. Vikram Bhatt and Mr. Nilesh Agarwal for efforts and time they spend to make me understand and implement streaming Naive Bayes algorithm and Hadoop MapReduce implementation for the same alogrithm respectively. Mr.Nilesh also helped me to understand how many ways I can implement the same algorithm in Hadoop through his expertise in the subject.

References

Cohen, William. Naive Bayes and Map-Reduce. Technical report, 2017. URL `https://pdfs.semanticscholar.org/a397/eb310921897ef8a140668b623de618da7606.pdf`.

White, Tom. *Hadoop: The Definitive Guide*. O'Reilly

Media, Inc., 1st edition, 2009. ISBN 0596521979, 9780596521974.