DS222: MLLD 2018
Assignment 1

Rishi Hazra
Systems Engineering
14542
rishixtreme@gmail.com

Abstract

This assignment is based on implementation of Naive Bayes classifier on articles in both local and Map-Reduce mode. We then, compare the time taken to complete the prediction process.

1 Introduction

1.1 Dataset

We use DBPedia [1] for out prediction task. This dataset is available in three types, namely *very small, small & full*. The *very small* dataset has 10,497 documents in train file, 2997 in validation file and 1497 in test file. It has 49 labels. The *full* dataset has 2,14,997 train documents, 61,497 validation documents and 29,997 test documents. It has 50 labels.

1.2 Problem Statement

In this assignment, we will train a Naive Bayes classifier in both local and MapReduce mode.

- Local Naive Bayes: In this part of assignment, we will implement Naive Bayes in-memory. I use python for my implementation.
- In this part of assignment we will re-implement Naive Bayes on Hadoop MapReduce framework.

2 Naive Bayes classifier

In machine learning, naive Bayes classifiers are a family of simple *probabilistic classifiers* based on applying **Bayes' theorem** with strong **(naive) independence** assumptions between the features. Abstractly, naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector $s = (w_1, ..., w_n)$ representing some n words w_i in a sentence (independent variables), it assigns to this instance probabilities

$$p(L_k|w_1, ..., w_n)$$

for each of K possible outcomes or labels L_k.

The problem with the above formulation is that if the number of features n is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. We therefore reformulate the model to make it more tractable. Using **Bayes' theorem**, the conditional probability can be decomposed as,

$$p(L_k|s) = \frac{p(L_k)p(s|L_k)}{p(s)}$$

Thus, the above equation can be written as,

$$posterior = \frac{prior * likelihood}{evidence}$$

Using chain rule on the joint probability,

$$\begin{aligned} p(L_k|w_1, .., w_n) &= p(w_1|w_2, .., w_n, L_k)p(w_2, ..w_n, L_k) \\ &= p(L_k)p(w_1|L_k)p(w_2|L_k)p(w_3|L_k)... \\ &= p(L_k)\prod_{i=1}^{N} p(w_i|C_k) \end{aligned}$$

This means that under the above independence assumptions, the conditional distribution over the class variable C is:

$$p(L_k|w_1, ..., w_n) = \frac{1}{Z}p(L_k)\prod_{i=1}^{N} p(w_i|C_k)$$

where the evidence $Z = p(s) = \sum_k p(L_k)p(s|L_k)$ is a scaling factor dependent only on $w_1, ...w_n$ that is, a constant if the values of the feature variables are known.

2.1 Multinomial naive Bayes

If a given class and feature value never occur together in the training data, then the frequency-based probability estimate will be zero. This is problematic because it will wipe out all information in the other probabilities when they are multiplied. Therefore, it is often desirable to incorporate a small-sample correction in all probability estimates such that no probability is ever set to be exactly zero. This regularization method is called **Laplace Smoothing**.

2.2 Naive Bayes Algorithm

TrainMultinomialNB(C, D)

1. $V \longleftarrow$ ExtractVocabulary(D)
2. $N \longleftarrow CountDocs(D)$
3. for each $c \in C$
4. do $N_c \longleftarrow$ CountDocsInClass(D, c)
   - $prior[c] = \frac{N_c}{N}$
   - $text_c \longleftarrow$ ConcatTextOfAllDocsInClass(D, c)
   - for each $t \in V$
   - $T_ct \longleftarrow$ CountTokensOfTerm$(text_c, t)$
   - for each $t \in V$
   - do $condprob[t][c] \longleftarrow \frac{T_{ct}+1}{\sum_{t'}(T_{ct'}+1)}$
5. return $V, prior, condprob$

ApplyMultinomialNB$(C, V, prior, condprob, d)$

1. $W \longleftarrow$ ExtractTokensDoc(V, d)
2. for $eachc \in C$
3. do $score[c] \longleftarrow \log prior[c]$
   - for each $t \in W$
   - do $score[c] += \log condprob[t][c]$
4. return $argmax_{c \in C} score[c]$

To eliminate zeros, we use add-one or Laplace smoothing, which simply adds one to each count called **Laplace Smoothing**.

$$\hat{P}(t|c) = \frac{T_{ct}+1}{\sum_{t' \in V}(T_{ct'}+1)} = \frac{T_{ct}+1}{(\sum_{t' \in V} T_{ct'}) + B'}$$

2.3 Naive Bayes on Hadoop MapReduce framework

Hadoop streaming is a utility that comes with the Hadoop distribution. This utility allows us to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer. In this assignment, both the mapper and the reducer are python scripts that read the input from standard input and emit the output to standard output. The utility will create a Map/Reduce job, submit the job to an appropriate cluster, and monitor the progress of the job until it completes.

When a script is specified for mappers, each mapper task will launch the script as a separate process when the mapper is initialized. As the mapper task runs, it converts its inputs into lines and feed the lines to the standard input (STDIN) of the process. In the meantime, the mapper collects the line-oriented outputs from the standard output (STDOUT) of the process and converts each line into a key/value pair, which is collected as the output of the mapper.

When a script is specified for reducers, each reducer task will launch the script as a separate process, then the reducer is initialized. As the reducer task runs, it converts its input key/values pairs into lines and feeds the lines to the standard input (STDIN) of the process. In the meantime, the reducer collects the line-oriented outputs from the standard output (STDOUT) of the process, converts each line into a key/value pair, which is collected as the output of the reducer [2].

3 Approach

3.1 Map Reduce model description

Training Phase :

- **MR1** : The first map-reduce operation is used to preprocess the training data to split the documents into label and text. The text is further processed to remove html links, special symbols, digits, extra spaces and stop words. The words are converted to lower case which gives us a lower size of vocabulary to deal with. The mapper then generates key-value pair *((label, word), 1)* where *(label, word)* forms a composite key. The last line of the mapper displays the total vocabulary count for unique words. The re-

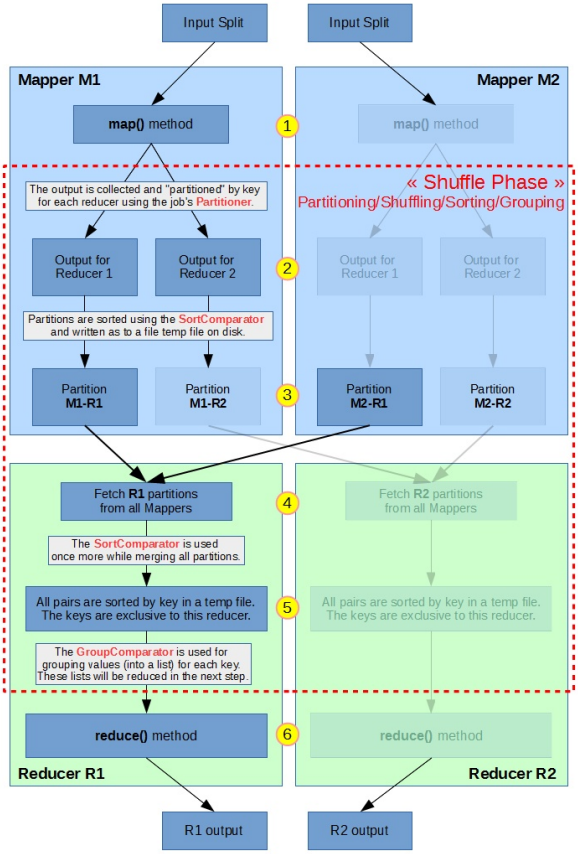

Figure 1: Hadoop streaming

ducer sums the count for similar words in a given class and stores it in this format: *((label, word), count)* with the last line being the vocab count.

- **MR2** : The second map-reduce takes the *MR1* output and counts the total number of words in each class. The mapper is very similar to a identity mapper which generates the output from the input with the *word* key discarded. Thus the mapper generated output is in the given form: *(label, count)*. The reducer groups and counts the total number of words in a given label.

- **MRconditional** : The map reduce works on the *MR1 & MR2* output to generate the logarithmic likelihood probabilities of each word given a label. The mapper is an identity mapper. The reducer uses hashing to divide the word count in each class (output of *MR1*) by the total number of words in that class (output of *MR2*). The output is generated in the following format: *((word, log likelihood), label)*.

- **MRprior** : This map-reduce operation works independently on the training data to generate the count of labels of each type. The mapper generates the output in the given format: *(label,1)*. The reducer then, sums the counts for each label to generate the data in the given format: *(label, label count)*. The last line of the reducer contains the *total count of all labels*.

- **MRprior2** : The map-reduce works on the *MRprior* output to obtain the prior logarithmic probabilities. The mapper is an identity mapper. The reducer then calculates the log probabilities by dividing the *label count* of each label by the *total count of each labels*.The output is generated in the following format: *(label, log prior probabilities)*.

Testing Phase : Only one stage of map-reduce is performed for testing.

- **MRtest** : This map-reduce operation does the combined task of calculating the posterior probabilities for each label of each document of the test data. The mapper takes three input files: *MRprior2 and MRconditional* as cache files and the testing data. The mapper preprocesses the testing data. It then generates a output format: *(label, test text)*. The reducer makes a hash function for each of the prior probabilities (output of*MRprior2*) and another hash function for likelihood of each word in each label (output of *MRconditional*). It then calculates the predicted label for each test document using Naive Bayes' algorithm. The accuracy is generated as output.

4 Experimental Results

Vocabulary Count (*very small*) = 506784
Number of training classes (*very small*) = 49
Vocabulary Count (*full*) = 506784
Number of training classes (*full*) = 50

4.1 Run Time

As one can observe from Table 1 , the time keeps falling as the number of reducers are increased as each reducer has to go through lesser data during streaming. We can also observe that if the increase

Table 1: Run Time

Method	Time
Local Naive Bayes'	5 min
Map Reduce (R=1)	7 min 47 sec
Map Reduce (R=2)	5 min 41 sec
Map Reduce (R=5)	4 min 49 sec
Map Reduce (R=8)	4 min 32 sec
Map Reduce (R=10)	4 min 28 sec

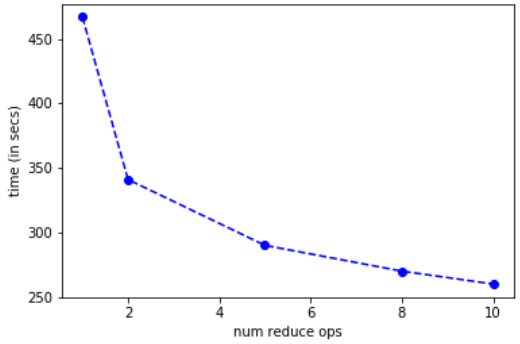

Figure 2: Time taken for different reduce ops

in number of reducers is small, the gain in computation cost may not overcome the communication cost of sending reducers to different nodes. It is also observed that the implementation on local machine is faster that distributed for lower number of reducers as all computations are done in memory. With the increase in number of map-reduce operations, the disk computation time is overshadowed by the parallel operations of the reducers.

4.2 Accuracy

Table 2: Accuracy

Method	Acc dev(%)	Acc test(%)
Local Naive Bayes'(very small)	85	84
Local Naive Bayes'(full)	70	69
Map Reduce (very small)	75	74
Map Reduce (full)	79	74

5 Acknowledgements

I would like to acknowledge Sai Suman for helping me debug the errors I was facing while streaming in Hadoop framework. Besides, I would like to thank Sourabh Balgi, Karan Malhotra, Chaikesh Chouragade, Shubham Bansal and Souvik Karmakar for helping me understand some integral concepts.

Table 3: Other Paramaters

Reducers	Read op	Write op	Shuffled maps	Merged
1	54	12	12	12
2	72	24	24	24
5	126	60	60	60
8	180	96	96	96
10	216	120	120	120

Here is the link to github repository.

References

[1] Auer, Sören and Bizer, Christian and Kobilarov, Georgi and Lehmann, Jens and Cyganiak, Richard and Ives, Zachary, "TDBpedia: A Nucleus for a Web of Open Data" roceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference.

[2] Dean, Jeffrey and Ghemawat, Sanjay, "MapReduce: Simplified Data Processing on Large Clusters" Commun. ACM, January 2008.