Assignment 1-Naive Bayes implementation in memory and using map-reduce framework

Machine Learning with large Datasets-2017
Sonali Singh, Mtech-SSA, Sr no. 13253

1. Local (in memory) Naive Bayes

Code was written in python and run on laptop with local memory. No GPU/cluster or any parallelism of code was used in this part.

1.1. Data Preparation

Punctuations: All english language standard punctuations and digits were removed .

Word filter on basis of length: All words with one or two english alphabets were removed in training.

Lower case: All words were converted to lower case with blank space removals.

Note: No stemming or stopword removal was done for training.

1.2. Training

In memory dictionaries: A total of fifty dictionaries were made in training process using python's default dictionary library. Each dictionary corresponded to a class label with words in class as key, and count of word in the respective class as values. for eg. Dictionary 1 with class 1 maintained count for :- (Y=y,W=w) number of times token w appears in a document with label y. An extra dictionary was maintained for counters:- (Y=y) for each label y the number of training instances of that class.

Trainable parameters: Total trainable parameters are 1088954. These parameters definition and training algorithm for obtaining trained parameters is as follows:- (Y=y) for each label y the number of training instances of that class (Y=*) here * means anything , so this is just the total number of training instances. (Y=y,W=w) number of times token w appears in a document with label y. (Y=y,W=*) total number of tokens for documents with label y. The learning algorithm just increments counters:

Algorithm::-
for each example y [w1,...,wN]:
increment (Y=y) by 1
increment (Y=*) by 1
for i=i to N:
increment (Y=y,W=wi) by 1
increment (Y=y,W=*) by N

Training Time: Training time for in memory naive bayes is **70.54 sec**

Training Accuracy: Training accuracy achieved for in memory naive bayes is **93.28 %**

1.3. Testing

The formula used for calculating log probabilities for a test case to belong to a particular class is:-
$ln(p(Y = y)) + \sum_w ln(p(W = w|Y = y))$
The label assigned for each test case is argmax over all probabilities for all classes.

Testing Time: Testing time for in memory naive bayes is **2103.76 sec**

Development Accuracy: Accuracy achieved for development data in memory naive bayes is **91.62 %**

Testing Accuracy: Testing accuracy achieved for in memory naive bayes is **91.17 %**

2. Naive Bayes Using Map-Reduce framework

Code was written in python and run on cluster using hadoop's map-reduce framework.

2.1. Data Preparation

Data preparation is same as mentioned in section 1.1 for in memory Naive Bayes. It is done in training mapper phase itself.

. Correspondence to: Sonali Singh <sonalis7722@gmail.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

Table 1. Training time in sec for Map - Reduce framework on training data.

REDUCERS	MAP	SHUFFLE	MERGE	REDUCE	TOTAL
2	275	48	8	128	459
4	142	33	4	65	244
6	95	23	3	43	164
8	78	35	2	33	148
10	59	16	2	26	103

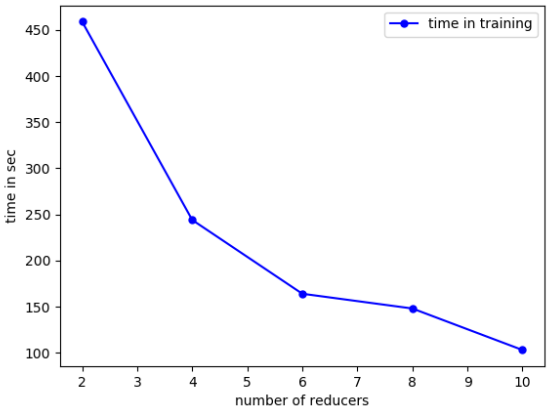

Figure 1. Plot indicating variation of total training time with varying number of reducers.

2.2. Training

One mapper and one reducer were used for training.

Mapper: Mapper was used to generate instances of following two types:-
(Y=y,W=w,1) Print token w appears in a document with label y with count 1.
(Y=y,1) Print for each label y the training instance.
for eg:
$[$"$YaleUniversityalumni$","$wrench$"$]$ 1
"!$ColumbiaUniversityalumni$" 1

Reducer: Aggregate all common keys and add their respective counts to get:- (Y=y) for each label y the number of training instances of that class.
(Y=y,W=w) number of times token w appears in a document with label y.
(Y=y,W=*) total number of tokens for documents with label y.
(Y=*) here * means anything , so this is just the total number of training instances.
for eg:-
$[$"$YaleUniversityalumni$","$wrench$"$]$ 126
"!$ColumbiaUniversityalumni$" 5967

Trainable parameters: Total trainable parameters are 1088954.

Training Time: The best training time obtained was with 10 reducers **103 sec** . See table 1 for detailed training time across different reducers. Also figure 1 plot shows variation of total training time with reducers.

Training Accuracy: Training accuracy achieved for map-reduce implementation of naive bayes is **93.28 %**

2.3. Testing

Three mappers and three reducers were used for testing phase. The explanations of each mapper and reducer are as below.

Input to Mapper1 was output of training phase reducer and test instances concatenated with line numbers as id.
Mapper1: Mapper was used to generate instances of following two types:-
(W=w,Y=y,count=c) Print token w appears in a document with label y with count c obtained from training phase reducers.
(W=w, $\sim$, id). Print for each line of test input word followed by $\sim$ and id. Here ids are line no. with true label.
for eg:
$acclaim\ YaleUniversityalumni$ 126
$acclaim \sim$ 1094 $Americanmalefilmactors$
$Americanmaletelevisionactors$

Reducer1: Aggregate all test case lines that is lines with $\sim$ with corresponding word and class counts. for eg:-
1094 $Americanmalefilmactors$
$Americanmaletelevisionactors \sim acclaim$
$YaleUniversityalumni$ 126

Input to mapper 2 is output of reducer1
Mapper2: Print lines of reducer 1 as it is. This is used to sort all lines by ids so that all same line numbers information can be aggregated together. for eg:
1094 $Americanmalefilmactors$
$Americanmaletelevisionactors \sim acclaim$
$YaleUniversityalumni$ 126

Table 2. Testing time in sec for Map - Reduce framework on testing data.

REDUCERS	MAP	SHUFFLE	MERGE	REDUCE	TOTAL
2	51	45	3	37	136
4	45	28	1	20	94
6	42	31	0	14	87
8	43	16	0	10	69
10	42	29	0	9	80

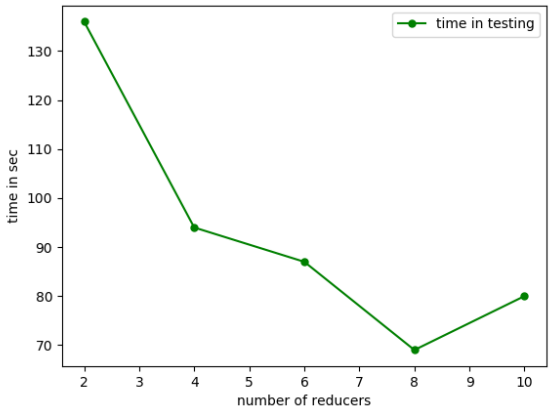

Figure 2. Plot indicating variation of total testing time with varying number of reducers.

1094 $Americanmalefilmactors$ $Americanmaletelevisionactors$ $\sim$ $abandoned$ $Americanfilms$ 28

Reducer2: Aggregate all test case line ids to generate all words and corresponding counts per test case id. for eg:-
1094 $Americanmalefilmactors$ $Americanmaletelevisionactors$ $\sim$ $acclaim$ $YaleUniversityalumni$ 126
$\sim$ $abandoned$ $Americanfilms$ 28

Input to mapper 3 is output of reducer2
Mapper3: Mapper was used to read each test instance calculate the maximum prbability class and print if prediction is correct or wrong. Note that fixed parameters that is (Y=y,W=*) total number of tokens for documents with label y.
and (Y=*) here * means anything , the total number of training instances , are passed through distributed cache. The cache file is a text file with hundred lines. The output of mapper is as below. for eg:
$correct$ 1
$error$ 1

Reducer3: Aggregate all correct and error test instance counts. for eg:-
$correct$ 2648
$error$ 27349

The net accuracy is calculated from reducer3's output using (correct-value/(corect-value+error-value))*100

Testing Time: The best testing time obtained was with 8 reducers i.e. **69 sec** . See table 2 for detailed testing time across different reducers. Also figure 2 shows variation of total testing time with reducers.

Development Accuracy: Accuracy achieved for development data in map-reduce naive bayes is **91.62 %**

Testing Accuracy: Testing accuracy achieved for map-reduce naive bayes is **91.17 %**

Plot: The schematics i.e Figure 3 show variation of overall time (training and testing both) as the number of reducers are varied.

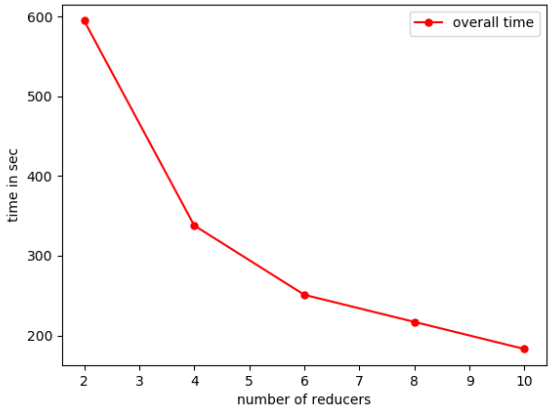

Figure 3. Plot indicating variation of overall (training and testing) time with varying number of reducers.

3. Conclusion

It is observed through experimentation that wall clock time is not linear with respect to number of reducers. In fact

reduction in time is drastic initially and decreases as number of reducers are increased. Also the best total training and testing time is **183 sec** with 10 reducers in map-reduce .However with execution on local machine total time is **2174.24 sec**. So we achieve a reduction in time for map-reduce implementation by **91.58 %** .In both the frameworks since the algorithm and preprocessing was same, except for implementation hence accuracies are same with test accuracy as **91.17 %**.