REPORT

Project

CS6240-Sec 2

By:

Abhay Kasturia

Rahul Verma

TABLE OF CONTENTS

Contents

HEADER

Class Number: CS6240 – Sec 2

HW number: Project

Name: Abhay Kasturia and Rahul Verma

Map-Reduce Source Code

Present in the zip file, with separate folders for each of the programs.

To run the codes unzip the codes folder and copy all the labelled source files to the labelled folder and the data to be predicted(unlabelled) in the unlabelled folder and make sure there is no output directory in the specific programs folder.

There is another configurable parameter which can be set in the makefile which is an integer and tells the number of models (Random Trees) to be built.

Run “make run” command inside the folder of the program needed to run and the output can then be checked in the output folder in the same directory.

Design Discussion and Pseudo-Code

1. Model and Parameters

We choose to go with the implementation of parallely trained Random Trees. These Random trees were trained on a subset of data with the same size as the size of the original data(Bootstrap Sample Z* of size N of training data). In this case we choose the row to be included in the tree at random - this will make the subsets to not have a few records and to have some duplicate rows. However the subsets of data will almost be of equal sizes. The parameters that we tuned was to keep the depth of the tree to be infinity by setting the MaxDepth to be zero, the number of attributes to randomly consider is square root of the number of attributes (25) , which is equal to 5. And the minimum number of node size is set to 1. These were set as per section 15.3 in The Elements of Statistical Learning by Trevor Hastie.

2. Pseudo-Code

The whole process is divided in three jobs:

1. PreProcessing and Building the model.
2. Validating the model
3. Prediction Job

1. Preprocessing and Building the model

This job consist of a single MapReduce Job. The mapper is responsible for preprocessing and sending the data in such a way that it is shuffled across the reducers. Also , it splits the data into 80-20 for Building the Model and Validation respectively.

Job1 : buildModel

// map receives one line from the labelled data

setup()

a. n = number of trees read from the config
b. init random functions and Multiple Outputs(mos)

map(line)

a. columns = split the line based on commas
b. filter only the records where the Primary checklist tag = 1 and the line is not the header line!
c. filter = picking only the required columns from the columns array
d. create a Sighting Detail object (new_attr) with all the required columns
e. rnd = random number from 0 to 9
f. if(rnd%10<8)
    // this part of the code ensure shuffling and bootstraps the data !
    a. Write to multiple output file train (0 , new_attr); // for training(keeping a copy of the training data)
    b. for(i=0 to n)
        i. emit(random number from 0 to n-1, new_attr) // the random function needs to work as such that for n number of runs it should give close to n unique values. This will ensure that the data is equally divided amongst the reducers.
g. else
    a. Write to multiple output file val (0 , new_attr); // for validation

partitioner(key , value , numPartitions)
return(key % numPartitions); // if number of trees are more than the number of reducers , then the load is distributed equally.

reduce(key , List of sight details)

a. create a new instances , called trainingSet // weka class Instances
b. for all sight details in the list
    a. Add the respective columns in the sight detail as the attributes to an instance and add the instance to the trainingSet // weka class Instance and attribute
c. Build a Random Tree classifier with the tuned parameters on this trainingSet and write it a local location as the model for that key. // note that the key would be in sequence from 0-(n-1) , so we will get n different models

2. Validating the model

This job consist of a single MAp only Job. The mapper is responsible to read all the models , load it in memory (as each model is not more than 500kb , this is a feasible option). For each of the sighting data we predict the values using our n models and then perform a Voting for the final prediction.

Job : valModel

// map receives one line from the formatted validation data

setup()

c. n = number of trees read from the config
d. init two counters correct=0,incorrect=0;
e. for(i=0 to n)
    i. read classifier number “i” and add it to the list of classifiers

map(Dummy key , Sight Detail)

h. init two counters t=0,f=0
i. create a new instance for the Sight Detail
j. Add the instance to a dataset
k. for(all classifiers in the list)
    a. if classifierDistributionforInstance is 1
        i. t++
    b. else
        i. f++;
l. if(t>f)
    a. emit(SightDetail.sampleID + “ , 1” , Dummy)
    b. if(SightDetail.getRed_Bird() = true) // actual value
        i. correct++;
    c. else
        i. incorrect++;
m. else
    a. emit(SightDetail.sampleID + “ , 0” , Dummy)
    b. if(SightDetail.getRed_Bird() = false) // actual value
        i. correct++;
    c. else
        i. incorrect++;

cleanup()

increment hadoop counter “correct” by the value of the local correct counter and same for the incorrect counter.

These counters can then be used in the driver program to calculate the efficiency by the formula correct/(correct+incorrect).

3. Testing the model (Total Order Partitioner)

The unlabeled Data set is Sequentially processed and integer value from 0 is appended to its last column to partition the data in the later steps and this file is saved (Sequential Step).Thies Sequential jobs also counts and save the number of unlabeled records in config as “unlabeledSize”.Total numbers of Reducers is 10.

Job : test

//Map

1. mapper(object,Text)
2. Convert Text to SightingDetail

3. make the appended integer as key and SightingDetail Value and write

//Partitioner

1. Setup()

   Receives value of Unlabeled Data from config and makes its 1/10 value(oneTenthSize).

2. Partition()

   Divides the Key by oneTenthSize and returns.

// Reduce receives one line from the formatted testing Unlabelled data

setup()

f. n = number of trees read from the config
g. init two counters correct=0,incorrect=0;
h. for(i=0 to n)
    i. read classifier number “i” and add it to the list of classifiers

map(Dummy key , Sight Detail)

n. init two counters t=0,f=0
o. create a new instance for the Sight Detail
p. Add the instance to a dataset
q. for(all classifiers in the list)
    a. if classifierDistributionforInstance is 1
        i. t++
    b. else
        i. f++;
r. if(t>f)
    a. emit(SightDetail.sampleID + “ , 1” , Dummy)

s. else
    a. emit(SightDetail.sampleID + “ , 0” , Dummy)

3. PreProcessing - What and Why?

In the preprocessing mapper we eliminate the columns we felt were irrelevant which included everything except the longitude , latitude , time , day , month , year , elevation and nearby water bodies data. We also converted the values of the Red bird occurrences with a true or a false. Also all the records with the primary checklist flag as not 1 were discarded as they were just duplicate reportings from the members of an observer group.

This was done to efficiently transmit the intermediate data and to keep only the relevant information in the attributes , to improve the accuracy of the system.

4. Salient Features

a. Randomness and why it is good ?

As per the following sources using bagging/bootstraping on a group of Random Trees is similar to creating a Random Forest as Random Forest is just a bagged version of decision trees except that at each split we only select 'm' randomly chosen attributes.

Random forest achieves a lower test error solely by variance reduction. Therefore increasing the number of trees in the ensemble won't have any effect on the bias of your model. Higher number of trees will only reduce the variance of your model. Moreover you can achieve a higher variance reduction by reducing the correlation between trees in the ensemble. This is the reason why we randomly select 'm' attributes at each split because it will introduce some randomness in to the ensemble and reduce the correlation between trees. Hence 'm' is the major attribute to be tuned in a random forest ensemble.

1) smaller the value of m will reduce the variance of the ensemble but will also increase the bias of an individual tree in the ensemble. 2) the value of m also depends on the ratio of noisy variables to important variables in our data set. For us since we have filtered a lot of noisy variables , the probability of choosing an important variable at a split thus affecting our model i a tad bit higher.

Source : https://www.youtube.com/watch?v=3kYujfDgmNk , http://statweb.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf , several other links on quora , stackoverflow and reddit.

b. Compared to other models

Compared to other Model we choose this because in highly unstable Data like ours, where change in one value can change the result Random trees are the best fit. The Accuracies compared to other models are a Proof.Bagging adds to this efficiency.

c. Efficiency

PreProcessing is Parallel and Scalable as it is Implemented in the Mapper Task.

Modeling is Parallel to a limit of 10 Reducers at a time to maintain the Accuracy of Data.(10 Models Prepared in Parallel)

Validation is also Scalable to any number of Reducers and Processing Power.(because Models is fetched and read in the Setup which a small task)

Testing could also have be Parallel given that we don't have to maintain the order of given Data.

d. Accuracy

Accuracy RoadMap

Single NaveBayes Weka Model : 67%

Single Predefined Mboost Weka Pre-Defined: 74%

Single Random Tree on 1/10th of 80% Training Data :75%

n Random Trees on 1/n of 80% Training Data: 81%(n=10) 78%(n=100)

n Random Trees on 1/n of 80% Training Data with Bagging :83.7% (n=10)

Final Method : Random Trees on 1/10 of 80% Training Data with Bagging :83.7%

e. Scale Up with respect to a Sequential Program

We have Implemented Random Forest Like Implementation in parallel where we model approx 10 Random Trees (For highly unstable Data) and then Vote among these Models.(

So Ensembled Method (Random Model) in Parallel which would have taken a lot of space in sequential.

Time wise it is faster because because for the number of records are tested in Parallel

f. Voting vs Average for Ensembling

As explained in the same book referenced above , when Random forests are used for classification, a random forest obtains a class vote from each tree, and then can classifies using majority vote. When used for regression, the predictions from each tree at a target point x are simply averaged.