CS6240 Parallel Data Processing in MapReduce

Final Project - NC Tracer Report

Yang Cai, Jingchao Cao, Li Xie

Introduction

We use Logistic Regression, Random Forest, Naive Bayes together as a bagging model. For each model, we explore different parameter settings and choose the best one with highest accuracy. Then we train three models together with best parameters.

For prediction, we use three models to vote for each record to determine whether it's foreground or background.

Model and Parameter Selection and Speed-Up Discussion

All the following experiments for same model discussion are based on same set of training and validation data, being sampled from image1,2,3 and image4 respectively, and also based on same set of parameters. For speed-up, as data cannot fit into single worker's memory, we use 5 machines runtime as baseline.

Logistic Regression

Speed-Up

Speed-up result is shown as following. Super-linear speed-up is observed, which should be related to cache. However, this is not observed in other models. Thus, this should also be related to the logistic regression model's running mechanism and amount of intermediate results. The reason also can be failure by overload of data of some of the machines during 5 workers configuration, which increase the total runtime.

# of workers	5	10	15
Runtime	53mins	13mins	7mins
Speedup	1.0	4.1	7.6

Parameters Exploration

<u>Maximum iteration:</u> The model reaches highest accuracy between iteration of 8 and 10, and stays stable after 20.

Regularization: The model reaches highest accuracy when reg param is 0.2, after 0.2, accuracy keeps decreasing.

Threshold: As a result, we choose to use (10, 0.2, 0.18) for (maximum iteration, regularization, threshold).

Random Forest

Speed-Up

The speed-up result is shown in following table. The result is basically same as the theoretical speed-up, which is consistent with our expectation that the perfect load-balance is achieved by MLlib training model, as Spark distributes training data and single model trains each part balanced and in parallel.

# of workers	5	10	15
Runtime	30mins	12mins	9mins
Speedup	1.0	2.5	3.3

Parameters Exploration

maxBins & numTrees: The maxBins and numTrees parameter doesn't have much effect on the accuracy. We tried maxBins = 5, 10, 20 and 32, numTrees = 5, 10, 15, 20, 25 and the accuracies are basically the same, which is around 0.9935646

maxDepth: The larger maxDepth will increase the accuracy, however the improvement is very limited. As we are exploring this parameter, each time we increase the maxDepth by 2, the accuracy increase about 0.000001. However, the computation time increases exponentially as the maxDepth increase. Considering the tradeoff between efficiency and accuracy, the maxDepth between 10 and 14 is acceptable.

maxDepth	5	8	10	12	14
accuracy	0.993564636	0.993565658	0.993566680	0.993570769	0.993571791

Thresholds: For thresholds array [a, 1 - a], when a is between 0.0 - 0.5, the accuracy are above than 0.99. And when a =~ 0.5, the accuracy reaches the maximum. If a keeps increasing, the accuracy will decrease sharply.

thresholds	[0.01, 0.99]	[0.1, 0.9]	[0.5, 0.5]	[0.9, 0.1]	[0.99, 0.01]
accuracy	0.99356770	0.99359836	0.99625387	0.98119373	0.93786643

Gradient Boosted Tree

We also tested GBT model provided by MLlib, and the accuracy is improved dramatically. However, it is very time consuming and resource consuming. For 600M training dataset, it took 3hrs with maxDepth 5 and maxIter 90. For 6G training dataset, out of memory error occurs on 18 workers, with maxDepth 15 and maxIter 90. Thus, we didn't use this model in our final bagging.

Naïve Bayes

Speed-Up

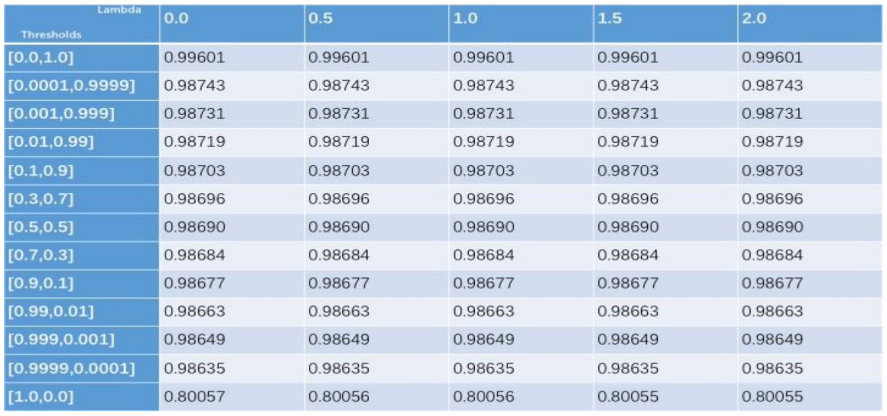
Ideal speed-up is observed by increasing the number of workers. As data is distributed over machines, training of single model is also distributed on machines corresponding to each partition of data. Thus, as worker number increases, the runtime decreases correspondingly.

# of workers	5	10	15
Runtime	12mins	6mins	4mins
Speedup	1.0	2.0	3.0

Parameter Exploration

<u>Smoothing (lambda):</u> lambda is used to set default prob. for element not appearing in training data. λ and #OfClass will be added to numerator and denominator of p(label=0) and p(label=1). When λ=1, it is Laplace smoothing and maintain total p=1. As lambda increases from 0 to 2 (= #OfClass), there is no changes over accuracy, there are 2 possible reasons. Firstly, as training data is large and element value range is limited, there may not be any fail-prob case. Secondly, as only balanced default prob. is supported, changes over lambda didn't change the predict ratio, and thus affected accuracy.
<u>Thresholds:</u> Thresholds can adjust the prob. of each class. The class with largest value (originalProb./classThres) is predicted. The best param is [0,1], which balances the prob. of label 0:1.

Accuracy results corresponding to different Lambda and Thresholds combinations is as following:



Lambda Thresholds	0.0	0.5	1.0	1.5	2.0
[0.0,1.0]	0.99601	0.99601	0.99601	0.99601	0.99601
[0.0001,0.9999]	0.98743	0.98743	0.98743	0.98743	0.98743
[0.001,0.999]	0.98731	0.98731	0.98731	0.98731	0.98731
[0.01,0.99]	0.98719	0.98719	0.98719	0.98719	0.98719
[0.1,0.9]	0.98703	0.98703	0.98703	0.98703	0.98703
[0.3,0.7]	0.98696	0.98696	0.98696	0.98696	0.98696
[0.5,0.5]	0.98690	0.98690	0.98690	0.98690	0.98690
[0.7,0.3]	0.98684	0.98684	0.98684	0.98684	0.98684
[0.9,0.1]	0.98677	0.98677	0.98677	0.98677	0.98677
[0.99,0.01]	0.98663	0.98663	0.98663	0.98663	0.98663
[0.999,0.001]	0.98649	0.98649	0.98649	0.98649	0.98649
[0.9999,0.0001]	0.98635	0.98635	0.98635	0.98635	0.98635
[1.0,0.0]	0.80057	0.80056	0.80056	0.80055	0.80055

Pseudo-Code

Preprocess and Induction

```
Object Train {
  Main(args) {
    // Pre-process (NOTE: rotation & mirror are in Induction section)
    trainingData = sc.textFile(image1.csv,image2.csv,image3.csv,image6.csv)
                   .map(toTrainRecord).toDS().sample(0.6).repartition(400)
    validationData = sc.textFile(image4.csv).map(toTrainRecord).toDS()
    // Induction
    modelNames = ["LR","RF","BY"]
    modelNames.map(modeName =>
      // construct (ModelEstimator, ParamGrid, TrainingData)
      metaData = {
        if (modelName == "LR") {
          diversifiedData = trainingData.map(mirrorAndRotate(0,0))
          lr = new LogisticRegression()
          paramGrid = new ParamGridBuilder()
            .addGrid(lr.maxIter, Array[MaxIter])
            .addGrid(lr.regParam, Array[RegParam])
            .addGrid(lr.threshold, Array[Threshold])
            .build()
          (lr, paramGrid, diversifiedData)
        } else if (modelName == "RF") {
          diversifiedData = trainingData.map(mirrorAndRotate(0,270))
          rf = new RandomForestClassifier()
          paramGrid = new ParamGridBuilder()
            .addGrid(rf.maxBins, Array[MaxBins])
            .addGrid(rf.numTrees, Array[NumTrees])
            .addGrid(rf.maxDepth, Array[MaxDepth])
            .addGrid(rf.thresholds, Array[Thresholds])
            .build()
          (rf, paramGrid, diversifiedData)
        } else { //modelName == "BY"
          diversifiedData = trainingData.map(mirrorAndRotate(180,180))
          by = new NaiveBayes()
          paramGrid = new ParamGridBuilder()
            .addGrid(by.smoothing, Array[Smoothing])
            .addGrid(by.thresholds, Array[Thresholds])
            .build()
          (by, paramGrid, diversifiedData)
        }
      }
      bestModel = validateToFindBestModel(metaData, validationData)
      bestMode.save()
    )
  }
}
```

Preprocess and Deduction

```
Object Predict {
  Main(args) {
    // Pre-process
    predictionData = sc.textFile(image5.csv).zipWithIndex().map(toIndexedRecord).toDS()
    // Deduction
    modelNames = ["LR","RF","BY"]
    results = modelNames
    .map(modelName =>
      model = loadModel(modelName)
      model.transform(predictionData).withColumnRenamed("prediction", modelName))
    .reduce((d1, d2) => d1.join(d2, "index"))
    .map{r =>
      votes = modelNames.map(modelName => r.getCol(modelName).reduce(_+_)
      predictLabel = if(votes > modelNames.size()/2) 1 else 0
      PredictRecord(r.getCol("features"), predictLabel)}
    results.save()
  }
}
```

Design Discussion

Preprocess

- If you performed any pre-processing, summarize what you did and why you did it?

We use image 1, 2, 3, 6 for training, and image 4 for validation.

If we use all the 25GB data for training, it takes long time to finish and tend to cause Out-of-Memory error. Also, we used different sizes of input for training, and found that the accuracy doesn't vary a lot between different sizes of input. Thus, we sample ~10GB training data from all the images, and for each model we transform the data by rotation and mirroring.

Induction

Induction process loops over models by array of model names. And for each of the model, it firstly creates a set of unique training data by applying rotation and mirror over the original training data to increase diversity. Then, it generates corresponding model classifier and parameter grid for the model. Then group data, classifier, and param grid as 3 element tuple. Then, for each kind of model, every param combination is trained on training data, and is evaluated on the validation data. Finally, the best model for each model type is saved for future prediction.

Take Logistic Regression as example:

- How many tasks are created during each stage of the model training process?

There are 400 partitions and thus tasks during both local and cloud running of training job on full dataset. There will be overload risk if the partition number is too low, as there will be 2GB limit for each partition on EMR.

- Is data being shuffled?

Yes. Model was firstly trained on each partition of training dataset, and there is no shuffling of data. After training on each partition is done, data is being shuffled during following treeAggregate job.

- How many iterations are executed during model training (for methods that have multiple iterations)?

For Logistic Regression, there is 10 times of iterations. While for Naive Bayes and Random Forest, there is no iterations.

Deduction

- How did changes of parameters controlling partitioning affect the running time?

At the beginning, as number of partitions increases, the running time will decrease, but there is a limit when number of partitions keeps increasing, the running time stays the same.

For this project, we are only using three models for bagging, so number of models is small. And our training data cannot fit into single worker's memory. So it's better that we partition by data. And in Spark, MLlib algorithms will train a subset of input data, and aggregate their results together to get the final model. It can achieve good speedup as long as we have enough partitions.

Result Discussion

Induction

All the following data are based on training on image 1 2 3 and 6, with three different kind of models (Logistic Regression, Random Forest, and Naive Bayes), each with one set of best parameters.

The speed-up result is shown as following. As logistic regression experiences some superlinear speed-up, the total speed-up is increased by it, and consistent with the sum of the runtime of single model with corresponding configuration, which means different models ran in sequential.

# of workers	5	10	15
Runtime	98mins	34mins	19mins
Speedup	1.0	2.9	5.2

Deduction

The speed-up result is shown as following. Near-ideal speed-up is observed, as to-be-predict data is partitioned and distributed over workers, and each work can access all models for bagging, and amount of data is large, the ideal speed-up is reasonable.

# of workers	5	10	15
Runtime	8mins	4mins	3mins
Speedup	1.0	2.0	2.7

Accuracy

We run training on 6G training data sampled from image1, 2, and 3, and validation data whole image 6. The following accuracy result is observed by running bagging prediction on whole image 4 dataset. The accuracy is higher than baseline.

Baseline (background # / total #)	0.993564
Accuracy	0.997153