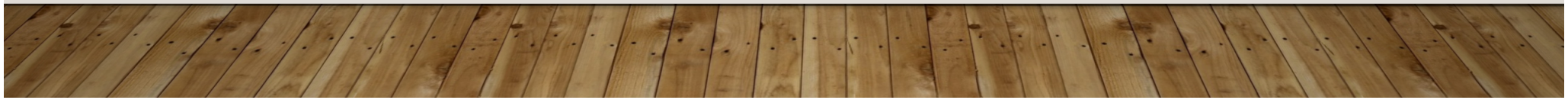
CS6240 Parallel Data Processing in MapReduce

Final Project - NC Tracer

Group Member: Yang Cai, Jingchao Cao, Li Xie



Outline

- Introduction
- Parameter Selection and Speed-Up Discussion
- Pseudo-Code
- Result Discussion
- Conclusion and Future Work

Introduction

- Training

  Bagging (Logistic Regression / Random Forest / Naïve Bayes)

  Each Model - Best parameter searched by paramGrid

- Prediction

  Each Pixel - 3 models votes for majority label

Model and Parameter Selection

- **Logistic Regression**

- Speed-Up

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

- Parameter Exploration

Maximum iteration: highest accuracy between iteration of 8 and 10, and stays stable after 20.

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

Threshold: (10, 0.2, 0.18) for (maximum iteration, regularization, threshold).

Model and Parameter Selection

- **Random Forest**

- Speed-Up

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

- Parameter Exploration

MaxBins & NumTrees: No much effect on the accuracy.

Thresholds:

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

MaxDepth:

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

Model and Parameter Selection

- **Naïve Bayes**

- Speed-Up

Ideal Speed-Up and balanced load by data distribution and parallel training

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

- Parameter Exploration

Smoothing Parameter λ:

Set default prob. for absence case.

No much effect on the accuracy.

Thresholds:

Adjust probability of classes by originalProb/classThreshold.

Best at [0.0,1.0]

Model and Parameter Selection

- **Naïve Bayes**

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**

- Training

```
// 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()
```

- Prediction

```
predictionData = sc.textFile(image5.csv)
                 .zipWithIndex()
                 .map(toIndexedRecord)
                 .toDS()
```

Pseudo-Code

- **Prediction**

```
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") {
      ...
    } else { //modelName == "BY"
      ...
    }
  }
  bestModel = validateToFindBestModel(metaData, validationData)
  bestMode.save()
)
```

Pseudo-Code

- **Prediction**

```
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()
```

Result Discussion

- **Speed-Up**

- Training

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

- Prediction

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

Result Discussion

- **Accuracy**

Training on 6G training data sampled from image1, 2, and 3,
Validation on whole image6
Prediction on whole image4

Evaludator metrics for both validation and prediction are based on ACCURACY

Baseline (background # / total #)	0.993564
Accuracy	0.997153

Conclusion and Future Work

- Conclusion
  - Spark & Mllib has good load balance on large dataset
  - We implemented a new feature to support different models ensembling, which provides good accuracy in this case.
- Future Work
  - More models to explore and include (SVM, GBT…)
  - Comparison with default ensembling (RF, GBT…)
  - Explore evaluation by AreaUnderROC