CS 6240: Final Project

Venkatesh Koka, Ramkishan Panthena, Karthik Subramanian

Github link: https://github.ccs.neu.edu/ramkishanp/cs6240

The initial phase is about learning about the input data. We read all the data to determine the class imbalance in the input. 5th image is used as test dataset. The data skew for 2nd image is very high among all the other images and closer to that of test image. So this image is used as the validation dataset and the rest of the images are used for training.

Rotations are done on the foreground records, so that we increase the number of data points belonging to foreground. Each input point has 3087 features and so the training the model is time intensive. To reduce the training time of model, we decided to filter out all the points that have pixel value greater than 20. This filter decreases the number of input features, so that less time taken to train on the entire dataset. Another method to reduce class imbalance is sampling the background data to achieve less skew between foreground and background points. We chose skew ratios of 1:1, 5:1, 20:1.

The baseline accuracy of any image dataset is determined by the ratio of majority class to the total size of data. We have used spark mllib for the ensemble libraries.

Experimenting with different classifiers is done to determine the model that have the highest accuracy for the input dataset. The following are the series of experiments

Note:: If all the 3087 features are used, the data will be further referred as dense representation else as sparse representation.

→ **Model used and Parameters Explored** :

Random Forest classifier :: TreeCount, MaxDepth, MaxBins.
5 decision trees are used with each having maximum depth as 30. The classifier parameter 'MaxBins' is set to 128.

Pre-processing :

Dense representation of data is used. This is used to get the initial accuracy without any reduction in input data.

Accuracy :

Using these parameters, we achieved an accuracy of 99.7204 that beats the baseline of 99.46 when tested on validation dataset i.e 2nd image.

Running Time :

Training on full data is a very time intensive process. We increased the number of trees from 5 to higher, but the process is taking forever to complete.

→ **Model used and Parameters Explored** :
Random Forest classifier :: TreeCount, MaxDepth.
5 decision trees are used with each having maximum depth as 30.
Data Pre-processing :
Sparse representation of data is used. This is used to improve the running time of the classifier
Accuracy :
Using these parameters, we achieved an accuracy of 99.7634 that beats the baseline of 99.46 when tested on validation dataset i.e 2nd image.
Running Time :
Training on sparse data made the classification be done faster compared to dense data. We increased the number of trees from 5 to 100 and increased the accuracy of the model.

→ **Model used and Parameters Explored** :
Random Forest classifier :: TreeCount, MaxDepth, SubSamplingRate.
100 decision trees are used with each having maximum depth as 30. SubSamplingRate of 0.3 is used so that only 30% of the total data is available for each tree.
Data Pre-processing :
Sparse representation of data is used. This is used to decrease the running time of prediction when compared to dense representation.
Accuracy :
Using these parameters, we achieved an accuracy of 99.7232 that beats the baseline of 99.46 when tested on validation dataset i.e 2nd image. Using the SubSamplingRate decreases the accuracy as the full data is not available for each tree.
Running Time :
Training on sparse data made the classification be done faster compared to dense data. We increased the number of trees from 5 to 100 and process.

→ **Model used and Parameters Explored** :
Random Forest classifier :: TreeCount, MaxDepth.
100 decision trees are used with each having maximum depth as 30.
Data Pre-processing :
Sparse representation of data is used. This is used to decrease the running time of prediction when compared to dense representation. Rotations of foreground class is used to increase the instances of this class as these are very sparse.
Accuracy :
Using these parameters, we achieved an accuracy of 99.5210 that beats the baseline of 99.46 when tested on validation dataset i.e 2nd image. The accuracy decreases because the model now starts predicting more background points as foreground points
Running Time :

Training on sparse data made the classification be done faster compared to dense data. We increased the number of trees from 5 to 100, if we increase the number of trees further the model is taking too much time.

→ **Model used and Parameters Explored** :
Random Forest classifier :: TreeCount, MaxDepth.
15 decision trees are used with each having maximum depth as 30. We have 3 different instances of classifier and 3 different dataset samples are fed to the classifiers.
Data Pre-processing :
The original dataset is sampled to form three different datasets having 1:1, 5:1 and 20:1 distribution of background to foreground points respectively. Sparse representation of data is used. This is used to decrease the running time of prediction when compared to dense representation.
Accuracy :
Using these parameters, and datasplits, we achieved an accuracy of 98.87, 99.54, 99.73 accuracy respectively that beats the baseline of 99.46 when tested on validation dataset i.e 2nd image. The accuracy decreases because the model now starts predicting more background points as foreground points
Running Time :
Having the data splits and running 3 classifiers took 1hr for training on one image data. To complete the training on all input datasets and doing the validation took 5hrs.

→ **Model used and Parameters Explored** :
Gradient Boosting classifier :: Iterations, MaxDepth.
100 iterations are set that means 100 decision trees are used. Maxdepth is set to 2 which is used to limit the number of nodes in the decision tree.
Data Pre-processing :
Sparse representation of data is used. This is used to decrease the running time of prediction when compared to dense representation.
Accuracy :
Using these parameters, we achieved an accuracy of 99.7383 accuracy that beats the baseline of 99.46 when tested on validation dataset i.e 2nd image.
Running Time :
The gradient boosting classifier took 1hr 20mins to go through training on 100 iterations and depth of 2.

→ **Model used and Parameters Explored** :
Naïve-Bayes classifier ::
Pseudo-code :
Data Pre-processing :

Sparse representation of data is used. This is used to decrease the running time of prediction when compared to dense representation.

Accuracy :

Using this model, we achieved an accuracy of 93.5234 accuracy respectively that is way less than the baseline of 99.46 when tested on validation dataset i.e 2nd image. But this model predicted the most true positives out of all the experiments, but this models also predicts more false negatives.

Running Time :

The running time of naïve-bayes classifier is 45 mins.

Final Run

→ **Models and Parameters used** :

Random Forest classifier :: TreeCount, MaxDepth.

100 decision trees are used with each having maximum depth as 30. We have split the entire dataset into 3 subsets and trained a separate model on each dataset.

1. Model 1: Random forest with 100 trees, 30 depth, on sparse dataset
2. Model 2: Boosted trees for 100 iterations, 2 depth on sparse dataset
3. Model 3: Boosted trees for 50 iterations, on 20:1 sampled sparse dataset

Pseudo-code :

For all the experiments done using Random forest Model, this pseudo-code is same for each one, but varying the respective parameters stated for each experiment.

Example PsuedoCode:

```
// This is for one random forest model.
// Read the input file
val inputTrain = sc.textFile(args(0))
            .map(line => line.split(","))


// Pre-process the input data by converting it to sparse representation and filtering out pixel values less than
20
def preprocWithoutNormalization(inputData: RDD[Array[Double]]): RDD[LabeledPoint] = {

  val preprocData = inputData.map(
          x => {x.zipWithIndex.map(_.swap).toSeq})

  val preprocData2 = preprocData.map(x => LabeledPoint(x.last._2,
Vectors.sparse(x.init.length, x.init.filter(v => v._2 > 20.0))))

     return preprocData2
}


trainSplits = trainData.split(0.33, 0.33, 0.33)
```

// Train a separate model on each split

Model1 = model.fit(trainSplit(0))

Model2 = model.fit(trainSplit(1))

Model3 = model.fit(trainSplit(2))

// Train a random forest model by providing the hyperparameters

```
def trainRF(trainData: RDD[LabeledPoint], strategy: Strategy, treeCount: Int, featureSubsetStrategy: String,
seed: Int): RandomForestModel = {
  val model = RandomForest.trainClassifier(trainData, strategy, treeCount, featureSubsetStrategy, seed)

  return model
}
```

// Train a Boosted trees model by providing the hyperparameters

```
val boostingStrategy = BoostingStrategy.defaultParams("Classification")
boostingStrategy.numIterations = 100
boostingStrategy.treeStrategy.numClasses = 2
boostingStrategy.treeStrategy.maxDepth = 2
boostingStrategy.treeStrategy.categoricalFeaturesInfo = Map[Int, Int]()
val modelbt1 = GradientBoostedTrees.train(trainData1, boostingStrategy)
```

// Make predictions for each test point

```
val labeledPredictionsrf = testData.map {labeledPoint =>
          val predictions = modelrf.predict(labeledPoint.features)
         (labeledPoint.label, predictions)}

val metricrf = new MultiClassMetrics(labeledPredictionsrf)

val accuracyrf = metricrf.accuracy
```

For this specific task, we partition the data and

No of Tasks: For rotations, we are filtering out the foreground images and generating new training records. Once data is generated, we are performing a union of it with the original file.

After merging, we shuffle and repartition the file before splitting it into different datasets for training multiple models.

Shuffling the data : For all cases where rotations are not being used, we are not shuffling the data. While using the rotations

How did changes of parameters controlling partitioning affect the running time? E.g., for bagging, was it better to partition the model file, the test data, or both?

We found it easier to partition the data and train a separate model on each partition. It was harder to partition the models and train them in parallel.

Data Pre-processing :

The original dataset is sampled to form three different datasets, the final one having 20:1 distribution of background to foreground points. Sparse representation of data is used. This is used to decrease the running time of prediction when compared to dense representation.

Accuracy :

Using these parameters, and datasplits, we achieved an accuracy of 98.763, 99.738, 99.733 accuracy respectively for three models, and an accuracy of 99.74 by using a majority vote that beats the baseline of 99.46 when tested on validation dataset i.e 2nd image. The accuracy increases because the model now predicts by using the majority vote from the three classifiers.

Running Time and Speedup:

Having the data splits and running 3 classifiers took 1hr for training on one image data. To complete the training on all input datasets and doing the validation took 5hrs 13 mins. Thus the speed up of our model would be approximately 5.