BIG DATA ANALYTICS AND APPLICATIONS

LAB ASSIGNMENT #2

Team Members

Shreyaa Sridhar - 21

Khushbu Kolhe - 9

Objective

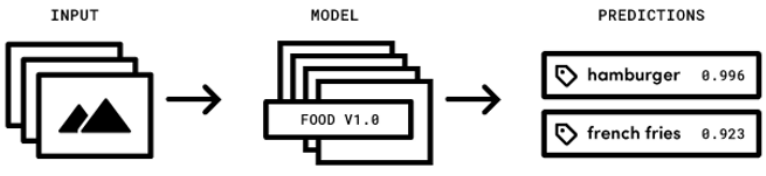
- To build a simple application to give the summary of a video by using Clarifai API. Using OpenImg Library to extract the key-frame images from the Clarifai API.
- To classify the images collected from videos or dataset(for your project) using the below classification algorithms

1. Naïve Bayes Model
2. Random Forest Model
3. Decision Trees Model

Features

Clarifai API

The Clarifai API offers image and video recognition as a service. You send inputs (an image or video) to the service and it returns predictions. The type of prediction is based on what model you run the input through. For example, if you run your input through the 'food' model, the predictions it returns will contain concepts that the 'food' model knows about. If you run your input through the 'color' model, it will return predictions about the dominant colors in your image.



Naive Bayes Model

Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes theorem with strong independence assumptions between the features.

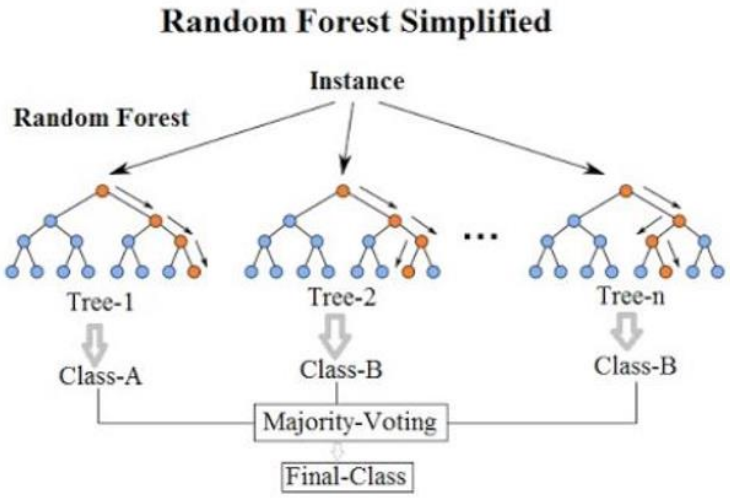
$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

Naive Bayes classification formula is

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \; p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

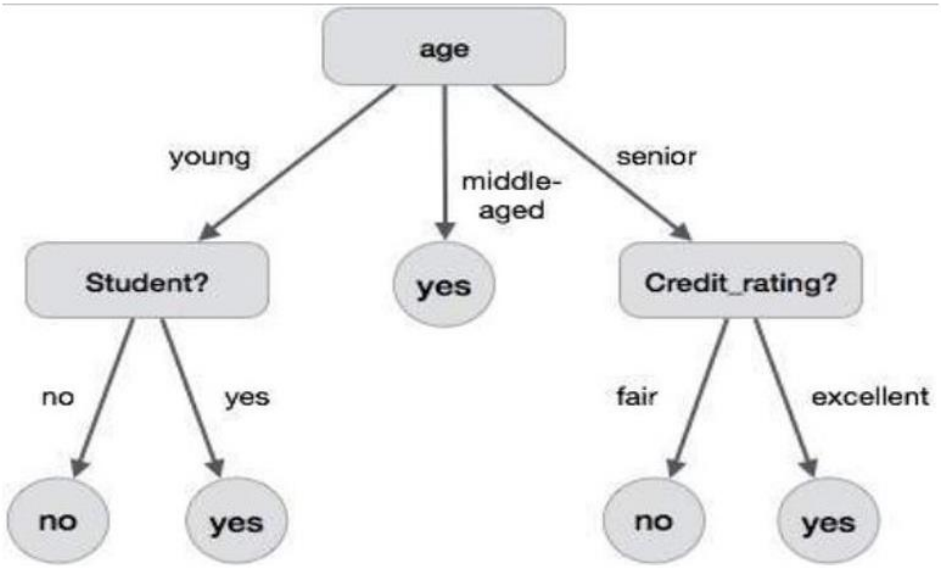
Random Forest Model

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.



Decision Tree Model

Decision tree learning uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modelling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.



Decision Tree Classification formula is

$$\overbrace{IG(T,a)}^{\text{Information Gain}} = \overbrace{H(T)}^{\text{Entropy(parent)}} - \overbrace{H(T|a)}^{\text{Weighted Sum of Entropy(Children)}}$$

Steps Involved

VIDEO ANNOTATION

1. Spark libraries are imported and spark is initialized.
2. Here we are analyzing video using Clarifai API and the video is split into keyframes and mainframes based on distinct frames.
3. Code ran successfully and output was generated.

KeyFrames and MainFrames generated

Output generated

Clarifai API has detected these based on the image.


image3
food
dish
hot
meal
dinner
lunch
no
person


image4
dish
fast
wrap
lunch
plate
epicure
lettuce
chicken
sauce
meal
delicious
dinner
onion


image5
homemade
traditional
tasty
refreshment
cake


image6
lunch
nature
health
traditional
vegetable
healthy
epicure

RANDOM FOREST MODEL

1. Spark libraries are imported and spark is initialized.
2. We have collected the input from **Food 101 dataset** .
3. Split data into training (70%) and test (30%) and Random forest function is applied.
4. Random forest Model is built and confusion matrix is obtained.
5. Accuracy of Random Forest Model is calculated.

Source Code

```
/**...*/
import java.nio.file.{Files, Paths}
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.tree.RandomForest
import org.apache.spark.mllib.tree.model.RandomForestModel
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}
import org.bytedeco.javacpp.opencv_highgui._
import scala.collection.mutable

object IPApp {
  System.setProperty("hadoop.home.dir","C:\\winutils");
  val featureVectorsCluster = new mutable.MutableList[String]

  val IMAGE_CATEGORIES = List("breakfast_burrito", "chicken_curry", "chocolate_cake",
    "french_fries", "garlic_bread", "hot_dog", "lasagna", "onion_rings", "pancakes")

 def generateRandomForestModel(sc: SparkContext): Unit = {
   if (Files.exists(Paths.get(IPSettings.RANDOM_FOREST_PATH))) {
     println(s"${IPSettings.RANDOM_FOREST_PATH} exists, skipping Random Forest model formation..")
     return
   }

   val data = sc.textFile(IPSettings.HISTOGRAM_PATH)
   val parsedData = data.map { line =>
     val parts = line.split(',')
     LabeledPoint(parts(0).toDouble, Vectors.dense(parts(1).split(' ').map(_.toDouble)))
   }

   // Split data into training (70%) and test (30%).
   val splits = parsedData.randomSplit(Array(0.7, 0.3), seed = 11L)
   val training = parsedData
   val test = splits(1)
   val numClasses = 9
   val categoricalFeaturesInfo = Map[Int, Int]()
   val maxBins = 100

   val numOfTrees = 4 to(10, 1)
   val strategies = List("all", "sqrt", "log2", "onethird")
   val maxDepths = 3 to(6, 1)
   val impurities = List("gini", "entropy")
```

```
var bestModel: Option[RandomForestModel] = None
var bestErr = 1.0
val bestParams = new mutable.HashMap[Any, Any]()
var bestnumTrees = 0
var bestFeatureSubSet = ""
var bestimpurity = ""
var bestmaxdepth = 0

numOfTrees.foreach(numTrees => {
  strategies.foreach(featureSubsetStrategy => {
    impurities.foreach(impurity => {
      maxDepths.foreach(maxDepth => {

        println("numTrees " + numTrees + " featureSubsetStrategy " + featureSubsetStrategy +
          " impurity " + impurity + " maxDepth " + maxDepth)

        val model = RandomForest.trainClassifier(training, numClasses, categoricalFeaturesInfo,
          numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins)

        val predictionAndLabel = test.map { point =>
          val prediction = model.predict(point.features)
          (point.label, prediction)
        }

        val testErr = predictionAndLabel.filter(r => r._1 != r._2).count.toDouble / test.count()
        println("Test Error = " + testErr)
        ModelEvaluation.evaluateModel(predictionAndLabel)

        if (testErr < bestErr) {
          bestErr = testErr
          bestModel = Some(model)
```

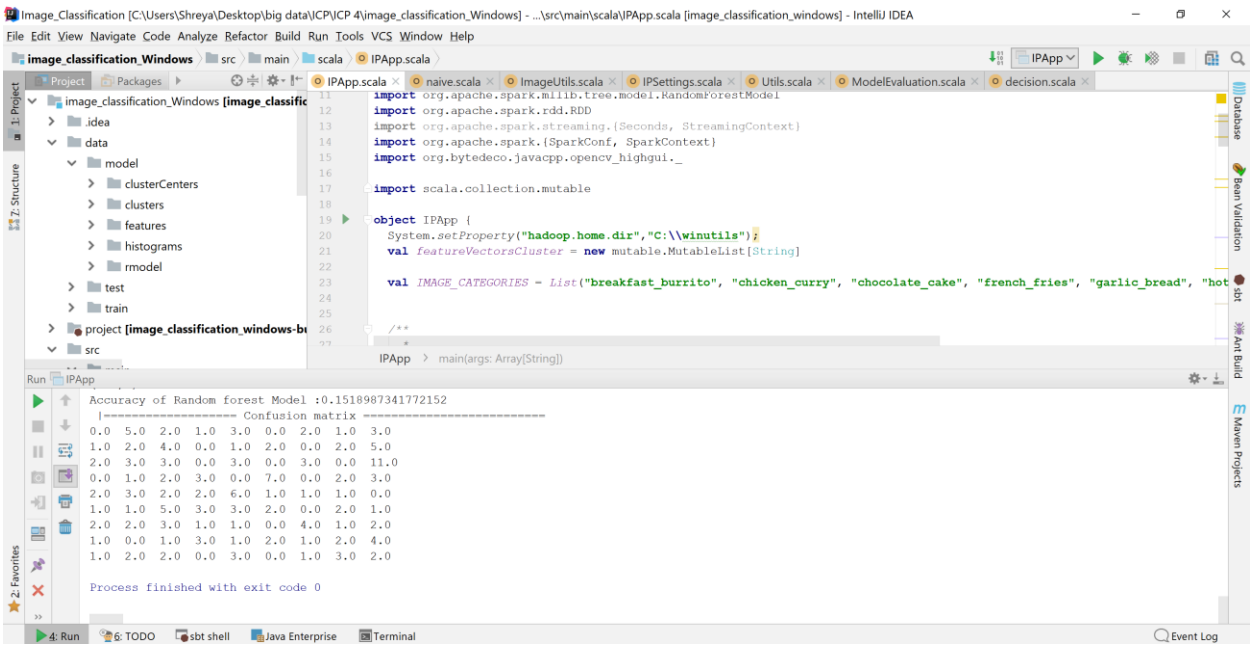
```
          bestParams.put("numTrees", numTrees)
          bestParams.put("featureSubsetStrategy", featureSubsetStrategy)
          bestParams.put("impurity", impurity)
          bestParams.put("maxDepth", maxDepth)

          bestFeatureSubSet = featureSubsetStrategy
          bestimpurity = impurity
          bestnumTrees = numTrees
          bestmaxdepth = maxDepth
        }
      })
    })
  })
})

println("Best Err " + bestErr)
println("Best params " + bestParams.toArray.mkString(" "))


val randomForestModel = RandomForest.trainClassifier(parsedData, numClasses,
  categoricalFeaturesInfo, bestnumTrees, bestFeatureSubSet, bestimpurity,
  bestmaxdepth, maxBins)


// Save and load model
randomForestModel.save(sc, IPSettings.RANDOM_FOREST_PATH)
println("Random Forest Model generated")
}
```

```
  def testImageClassification(sc: SparkContext) = {

    val model = KMeansModel.load(sc, IPSettings.KMEANS_PATH)
    val vocabulary = ImageUtils.vectorsToMat(model.clusterCenters)

    val path = "files/101_ObjectCategories/ant/image_0012.jpg"
    val desc = ImageUtils.bowDescriptors(path, vocabulary)

    val testImageMat = imread(path)
    imshow("Test Image", testImageMat)

    val histogram = ImageUtils.matToVector(desc)

    println("-- Histogram size : " + histogram.size)
    println(histogram.toArray.mkString(" "))

    val nbModel = RandomForestModel.load(sc, IPSettings.RANDOM_FOREST_PATH)
    //println(nbModel.labels.mkString(" "))

    val p = nbModel.predict(histogram)
    println(s"Predicting test image : " + IMAGE_CATEGORIES(p.toInt))

    waitKey(0)
  }
  def classifyImage(sc: SparkContext, path: String): Double = {

    val model = KMeansModel.load(sc, IPSettings.KMEANS_PATH)
    val vocabulary = ImageUtils.vectorsToMat(model.clusterCenters)
    val desc = ImageUtils.bowDescriptors(path, vocabulary)
    val histogram = ImageUtils.matToVector(desc)
    println("Histogram size : " + histogram.size)
    val RModel = RandomForestModel.load(sc, IPSettings.RANDOM_FOREST_PATH)
    val p = RModel.predict(histogram)
    p
  }

  def main(args: Array[String]) {
    val conf = new SparkConf()
      .setAppName(s"IPApp")
      .setMaster("local[*]")
      .set("spark.executor.memory", "6g")
      .set("spark.driver.memory", "6g")
    val sparkConf = new SparkConf().setAppName("SparkWordCount").setMaster("local[*]")
    val sc=new SparkContext(sparkConf)
    val images = sc.wholeTextFiles(s"${IPSettings.INPUT_DIR}/*/*.jpg")
    extractDescriptors(sc, images)
    kMeansCluster(sc)
    createHistogram(sc, images)
    generateRandomForestModel(sc)
    val testImages = sc.wholeTextFiles(s"${IPSettings.TEST_INPUT_DIR}/*/*.jpg")
    val testImagesArray = testImages.collect()
    var predictionLabels = List[String]()
    testImagesArray.foreach(f => {
      println(f._1)
      val splitStr = f._1.split("file:/")
```

```
    val predictedClass: Double = classifyImage(sc, splitStr(1))
    val segments = f._1.split("/")
    val cat = segments(segments.length - 2)
    val GivenClass = IMAGE_CATEGORIES.indexOf(cat)
    println(s"Predicting test image : " + cat + " as " + IMAGE_CATEGORIES(predictedClass.toInt))
    predictionLabels = predictedClass + ";" + GivenClass :: predictionLabels
  })

  val pLArray = predictionLabels.toArray

  predictionLabels.foreach(f => {
    val ff = f.split(";")
    println(ff(0), ff(1))
  })
  val predictionLabelsRDD = sc.parallelize(pLArray)


  val pRDD = predictionLabelsRDD.map(f => {
    val ff = f.split(";")
    (ff(0).toDouble, ff(1).toDouble)
  })
  val accuracy = 1.0 * pRDD.filter(x => x._1 == x._2).count() / testImages.count

  println("Accuracy of Random forest Model :"+accuracy)
  ModelEvaluation.evaluateModel(pRDD)

}
}
```

Random Forest Model generated

Code ran successfully and outputs were generated. Also the model was saved.



NAIVE BAYES MODEL

1. Spark libraries are imported and spark is initialized.
2. We have collected the input from **Food 101 dataset** .
3. Split data into training (70%) and test (30%) and Random forest function is applied.
4. Naive Bayes Model is built and confusion matrix is obtained.
5. Accuracy of Naive Bayes Model is calculated.

Source Code

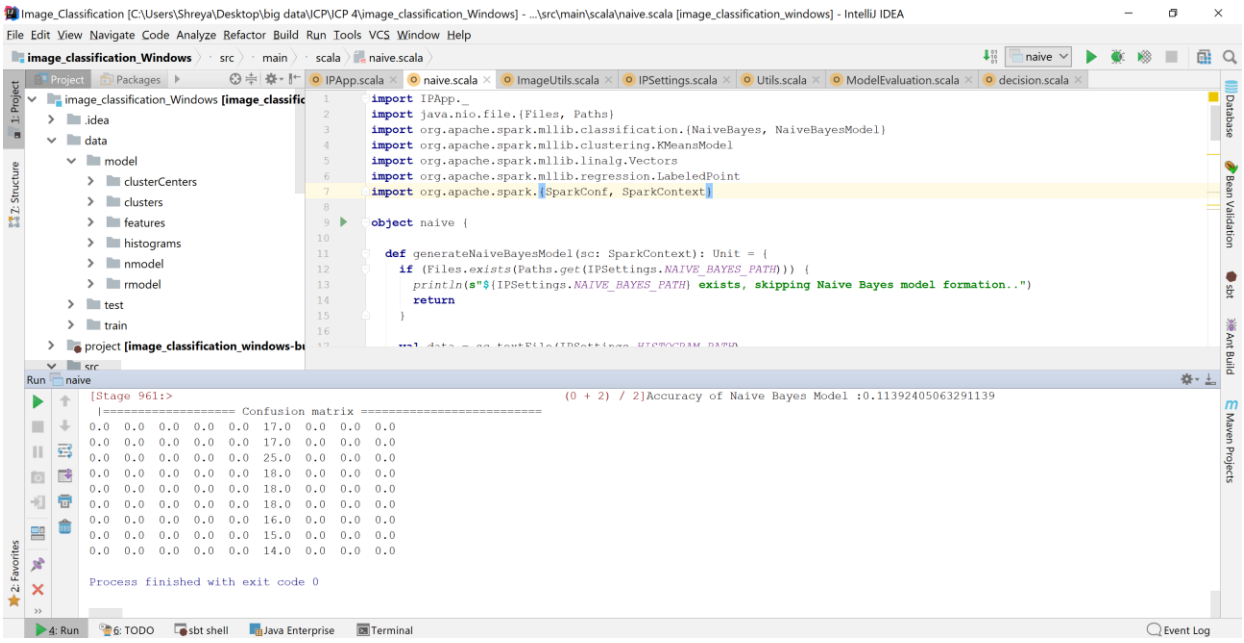
```
import IPApp._
import java.nio.file.{Files, Paths}
import org.apache.spark.mllib.classification.{NaiveBayes, NaiveBayesModel}
import org.apache.spark.mllib.clustering.KMeansModel
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.{SparkConf, SparkContext}

object naive {

  def generateNaiveBayesModel(sc: SparkContext): Unit = {
    if (Files.exists(Paths.get(IPSettings.NAIVE_BAYES_PATH))) {
      println(s"${IPSettings.NAIVE_BAYES_PATH} exists, skipping Naive Bayes model formation..")
      return
    }

    val data = sc.textFile(IPSettings.HISTOGRAM_PATH)
    val parsedData = data.map { line =>
      val parts = line.split(',')
      LabeledPoint(parts(0).toDouble, Vectors.dense(parts(1).split(' ').map(_.toDouble)))
    }

    val splits = parsedData.randomSplit(Array(0.7, 0.3), seed = 11L)
    print("splits size  = " + splits.size)
    val trainingData = splits(0)
    val testData = splits(1)

    val model = NaiveBayes.train(trainingData, lambda = 1.0, modelType = "multinomial")
```

```
  // Evaluate model on test instances and compute test error
  val labelAndPreds = testData.map { point =>
    val prediction = model.predict(point.features)
    (point.label, prediction)
  }
  val testErr = labelAndPreds.filter(r => r._1 != r._2).count().toDouble / testData.count()
  println("Test Error = " + testErr)
  println(model.modelType)

  // Save and load model
  model.save(sc, IPSettings.NAIVE_BAYES_PATH)
  println("Naive Bayes Model generated")
  val sameModel = NaiveBayesModel.load(sc, IPSettings.NAIVE_BAYES_PATH)
}


def main(args: Array[String]) {

  System.setProperty("hadoop.home.dir","C:\\winutils");

  val conf = new SparkConf()
    .setAppName(s"IPApp")
    .setMaster("local[*]")
    .set("spark.executor.memory", "6g")
    .set("spark.driver.memory", "6g")
  val sparkConf = new SparkConf().setAppName("SparkWordCount").setMaster("local[*]")

  val sc=new SparkContext(sparkConf)

  generateNaiveBayesModel(sc)
  val testImages = sc.wholeTextFiles(s"${IPSettings.TEST_INPUT_DIR}/*/*.jpg")
  val testImagesArray = testImages.collect()
  var predictionLabels = List[String]()
  testImagesArray.foreach(f => {
    println(f._1)
    val splitStr = f._1.split("file:/")
    val predictedClass: Double = NClassifyImage(sc, splitStr(1))
    val segments = f._1.split("/")
    val cat = segments(segments.length - 2)
    val GivenClass = IMAGE_CATEGORIES.indexOf(cat)
    println(s"Predicting test image : " + cat + " as " + IMAGE_CATEGORIES(predictedClass.toInt))
    predictionLabels = predictedClass + ";" + GivenClass :: predictionLabels
  })

  val pLArray = predictionLabels.toArray

  predictionLabels.foreach(f => {
    val ff = f.split(";")
    println(ff(0), ff(1))
  })
  val predictionLabelsRDD = sc.parallelize(pLArray)
  val pRDD = predictionLabelsRDD.map(f => {
    val ff = f.split(";")
    (ff(0).toDouble, ff(1).toDouble)
  })
  val accuracy = 1.0 * pRDD.filter(x => x._1 == x._2).count() / testImages.count

  println("Accuracy of Naive Bayes Model :" +accuracy)
  ModelEvaluation.evaluateModel(pRDD)
}
```

```
  def NClassifyImage(sc: SparkContext, path: String): Double = {

    val model = KMeansModel.load(sc, IPSettings.KMEANS_PATH)
    val vocabulary = ImageUtils.vectorsToMat(model.clusterCenters)

    val desc = ImageUtils.bowDescriptors(path, vocabulary)

    val histogram = ImageUtils.matToVector(desc)

    println("Histogram size : " + histogram.size)

    val nbModel = NaiveBayesModel.load(sc, IPSettings.NAIVE_BAYES_PATH)
    val p = nbModel.predict(histogram)
    p
  }
}
```

Naive Bayes Model generated

Code ran successfully and outputs were generated. Also the model was saved.



DECISION TREE MODEL

1. Spark libraries are imported and spark is initialized.
2. We have collected the input from **Food 101 dataset** .
3. Split data into training (70%) and test (30%) and Random forest function is applied.
4. Decision Tree Model is built and confusion matrix is obtained.
5. Accuracy of Decision Tree Model is calculated.

Source Code

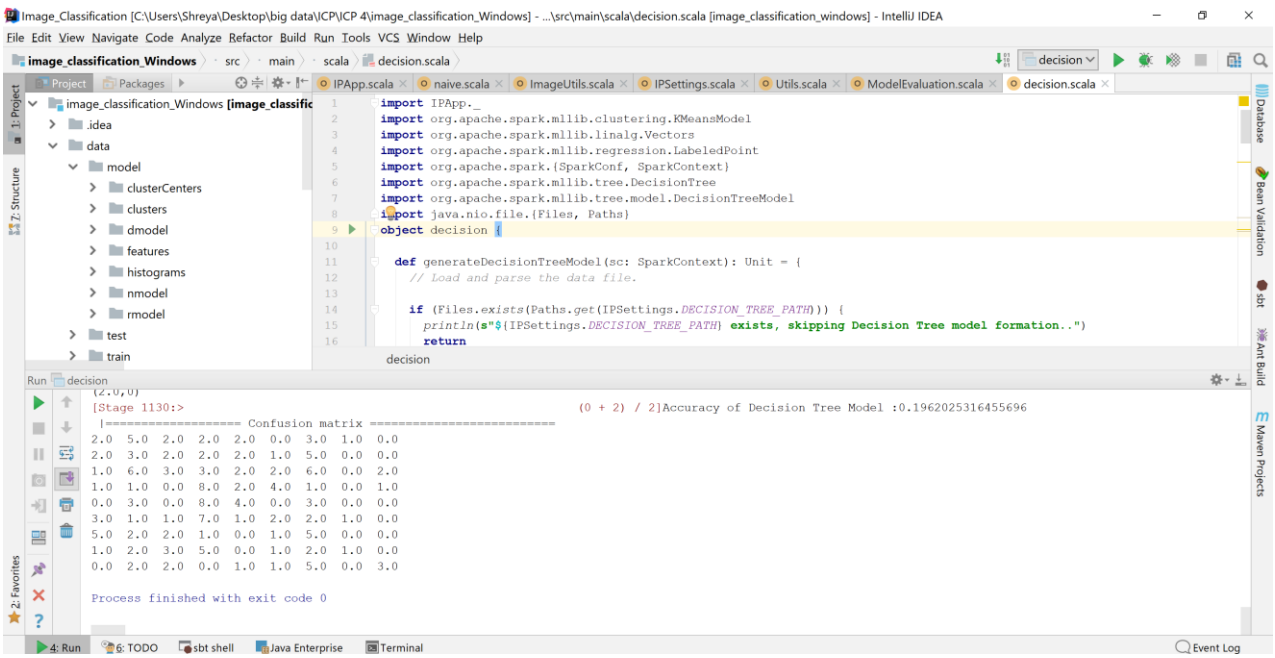
```
import IPApp._
import org.apache.spark.mllib.clustering.KMeansModel
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.model.DecisionTreeModel
import java.nio.file.{Files, Paths}
object decision {

  def generateDecisionTreeModel(sc: SparkContext): Unit = {
    // Load and parse the data file.

    if (Files.exists(Paths.get(IPSettings.DECISION_TREE_PATH))) {
      println(s"${IPSettings.DECISION_TREE_PATH} exists, skipping Decision Tree model formation..")
      return
    }

    val data = sc.textFile(IPSettings.HISTOGRAM_PATH)
    val parsedData = data.map { line =>
      val parts = line.split(',')
      LabeledPoint(parts(0).toDouble, Vectors.dense(parts(1).split(' ').map(_.toDouble)))
    }

    // Split data into training (70%) and test (30%).
    val splits = parsedData.randomSplit(Array(0.7, 0.3), seed = 11L)
    print("splits size  = " + splits.size)
    val trainingData = splits(0)
    val testData = splits(1)
```

```
    // Train a DecisionTree model.
    val numClasses = 9
    val categoricalFeaturesInfo = Map[Int, Int]()
    val impurity = "gini"
    val maxDepth = 5
    val maxBins = 32

    val model = DecisionTree.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo,
      impurity, maxDepth, maxBins)

    // Evaluate model on test instances and compute test error
    val labelAndPreds = testData.map { point =>
      val prediction = model.predict(point.features)
      (point.label, prediction)
    }
    val testErr = labelAndPreds.filter(r => r._1 != r._2).count().toDouble / testData.count()
    println("Test Error = " + testErr)
    println(model.toDebugString)

    // Save and load model
    model.save(sc, IPSettings.DECISION_TREE_PATH)
    println("Decision Tree Model generated")
    val sameModel = DecisionTreeModel.load(sc, IPSettings.DECISION_TREE_PATH)
  }

  def main(args: Array[String]) {

    System.setProperty("hadoop.home.dir","C:\\winutils");
    val conf = new SparkConf()
      .setAppName(s"IPApp")
      .setMaster("local[*]")
      .set("spark.executor.memory", "6g")
      .set("spark.driver.memory", "6g")
    val sparkConf = new SparkConf().setAppName("SparkWordCount").setMaster("local[*]")
    val sc=new SparkContext(sparkConf)
    generateDecisionTreeModel(sc)
    val testImages = sc.wholeTextFiles(s"${IPSettings.TEST_INPUT_DIR}/*/*.jpg")
    val testImagesArray = testImages.collect()
    var predictionLabels = List[String]()
    testImagesArray.foreach(f => {
      println(f._1)
      val splitStr = f._1.split("file:/")
      val predictedClass: Double = DClassifyImage(sc, splitStr(1))
      val segments = f._1.split("/")
      val cat = segments(segments.length - 2)
      val GivenClass = IMAGE_CATEGORIES.indexOf(cat)
      println(s"Predicting test image : " + cat + " as " + IMAGE_CATEGORIES(predictedClass.toInt))
      predictionLabels = predictedClass + ";" + GivenClass :: predictionLabels
    })

    val pLArray = predictionLabels.toArray
    predictionLabels.foreach(f => {
      val ff = f.split(";")
      println(ff(0), ff(1))
    })
    val predictionLabelsRDD = sc.parallelize(pLArray)
    val pRDD = predictionLabelsRDD.map(f => {
```

```
      val ff = f.split(";")
      (ff(0).toDouble, ff(1).toDouble)
    })
    val accuracy = 1.0 * pRDD.filter(x => x._1 == x._2).count() / testImages.count
    println("Accuracy of Decision Tree Model :" +accuracy)
    ModelEvaluation.evaluateModel(pRDD)
  }
  def DClassifyImage(sc: SparkContext, path: String): Double = {

    val model = KMeansModel.load(sc, IPSettings.KMEANS_PATH)
    val vocabulary = ImageUtils.vectorsToMat(model.clusterCenters)

    val desc = ImageUtils.bowDescriptors(path, vocabulary)

    val histogram = ImageUtils.matToVector(desc)

    println("Histogram size : " + histogram.size)

    val DTModel = DecisionTreeModel.load(sc, IPSettings.DECISION_TREE_PATH)

    val p = DTModel.predict(histogram)
    p
  }
}
```

Decision Tree Model generated

Code ran successfully and outputs were generated. Also the model was saved.



Summary

The accuracy of Random Forest, Naive Bayes and Decision Tree Models with respect to 9 classes considered are

	Random Forest Model	Naïve Bayes Model	Decision Tree Model
Accuracy with respect to 9 classes	0.151 (15.1%)	0.113 (11.3%)	0.196 (19.6%)

Based on our dataset we have received low accuracy for all models. But on comparing the values based on our data, Decision Tree model is the best model based on the accuracy. Naive Bayes seems to be the lowest accuracy model based on the dataset for 9 classes taken.

Configuration

- Intellij IDE
- Apache Spark(we have used the spark 2.2.1 version)

Source Code

Video Annotation

https://github.com/shreyaabadri/Big-Data-Analytics-and-Applications-Lab-Assignments/tree/master/LAB2/Source/Video%20Annotation

Image Classification

https://github.com/shreyaabadri/Big-Data-Analytics-and-Applications-Lab-Assignments/tree/master/LAB2/Source/Image%20Classification

Contribution

Shreyaa Sridhar (21) - 50 %

Khushbu Kolhe (9) - 50%

References

https://www.clarifai.com/developer/

https://spark.apache.org/docs/1.5.1/mllib-naive-bayes.html

https://spark.apache.org/docs/1.5.1/mllib-decision-tree.html

http://openimaj.org/