(Big Data Analytics Midterm)

Question1 –

a) Pre-processing steps considered are:-
b) Read the YearPerdictionFile.txt file and each sample in data has been normalized using L^2 normalizer and L Infinity Normalizer. L2 normalization is a vector norm.
c) Use the ChiSelector to select the different features in the Algorithms. As the doesn't allow more than 10000 column values, this Selector cannot be applied to Year PerdictionData.

Following are the Error:

```
// Chi test cannot be performed , because there are more than 10000 differnt
values in each column
        // Discretize data in 16 equal bins since ChiSqSelector requires
categorical features
        /*val discretizedData = data.map { line =>
          val parts = line.split(',')
          val myList = Array(parts(12), parts(78))
          LabeledPoint(parts(91).toDouble, Vectors.dense(myList.map(x =>
x.toDouble).toArray))
        }
        // Create ChiSqSelector that will select 50 features
        val selector = new ChiSqSelector(50)
        // Create ChiSqSelector model (selecting features)
        val transformer = selector.fit(discretizedData)
        // Filter the top 50 features from each feature vector
        val filteredData = discretizedData.map { line =>
          LabeledPoint(line.label, transformer.transform(line.features))
              }*/
```

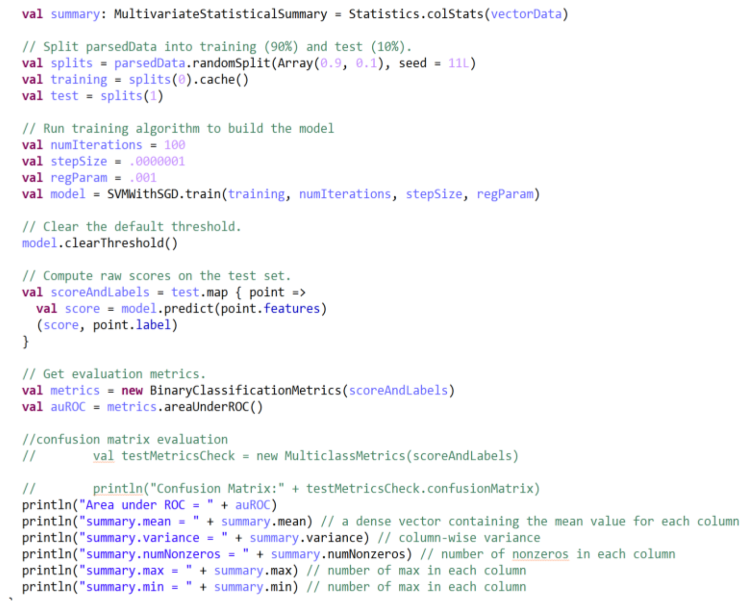
d) Result comparison – PCA(Principal Component Analysis) and SVD(Singular Value Decomposition)

   (Classification)

   PCA is used to feature reduction.

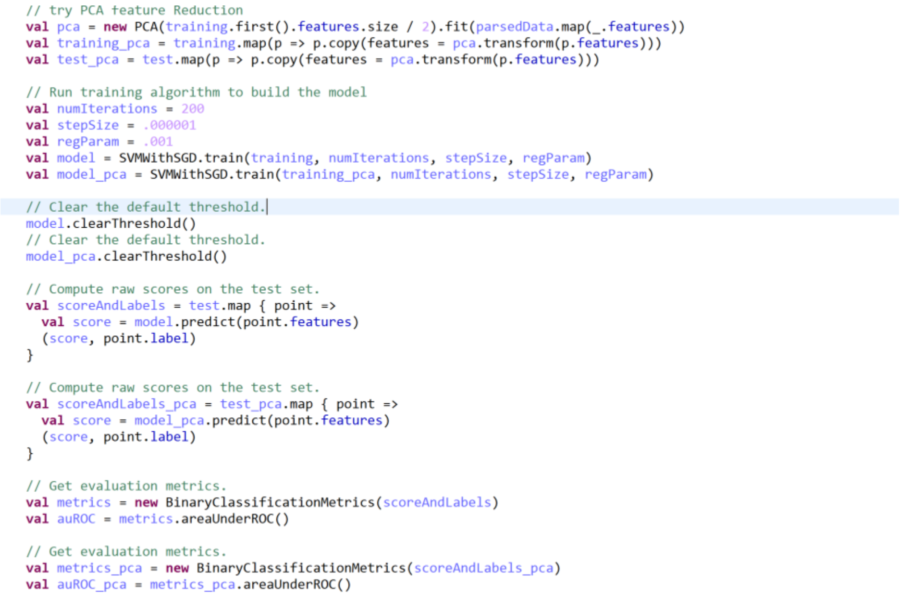
e) Used MultivariateStatisticalSummary in order to get the mean median , mode, sum values of each columns.
   Below is the code:
   val summary: MultivariateStatisticalSummary = Statistics.colStats(vectorData)

SVMwithSGD Code Implementation



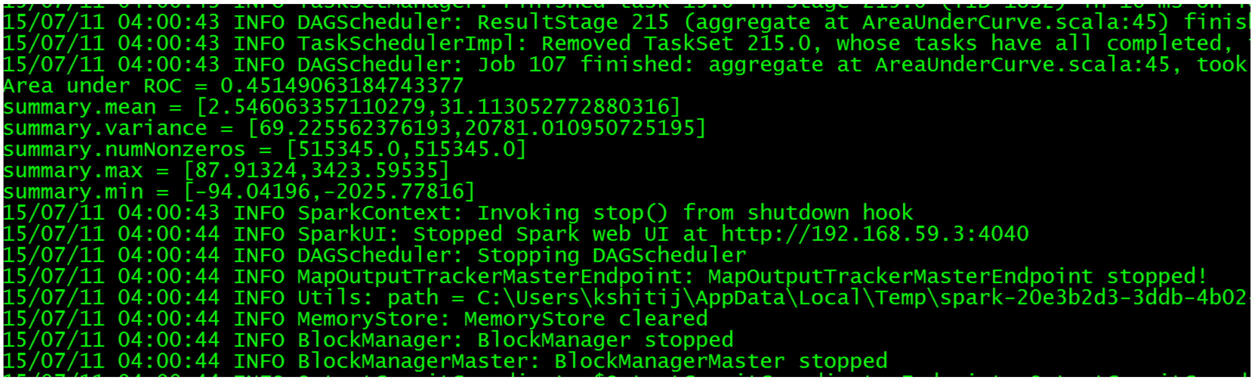
```
val summary: MultivariateStatisticalSummary = Statistics.colStats(vectorData)

// Split parsedData into training (90%) and test (10%).
val splits = parsedData.randomSplit(Array(0.9, 0.1), seed = 11L)
val training = splits(0).cache()
val test = splits(1)

// Run training algorithm to build the model
val numIterations = 100
val stepSize = .0000001
val regParam = .001
val model = SVMWithSGD.train(training, numIterations, stepSize, regParam)

// Clear the default threshold.
model.clearThreshold()

// Compute raw scores on the test set.
val scoreAndLabels = test.map { point =>
  val score = model.predict(point.features)
  (score, point.label)
}

// Get evaluation metrics.
val metrics = new BinaryClassificationMetrics(scoreAndLabels)
val auROC = metrics.areaUnderROC()

//confusion matrix evaluation
//        val testMetricsCheck = new MulticlassMetrics(scoreAndLabels)

//        println("Confusion Matrix:" + testMetricsCheck.confusionMatrix)
println("Area under ROC = " + auROC)
println("summary.mean = " + summary.mean) // a dense vector containing the mean value for each column
println("summary.variance = " + summary.variance) // column-wise variance
println("summary.numNonzeros = " + summary.numNonzeros) // number of nonzeros in each column
println("summary.max = " + summary.max) // number of max in each column
println("summary.min = " + summary.min) // number of max in each column
```

SVMwithSGD (PCA) code Implementation



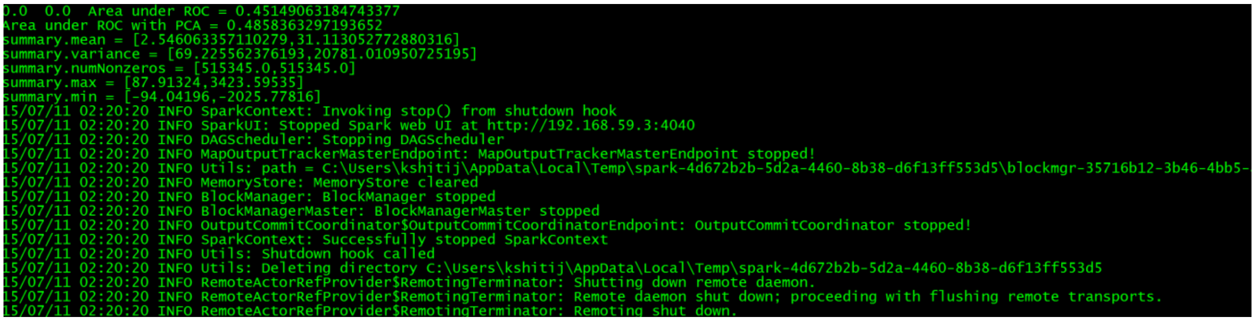
```
// try PCA feature Reduction
val pca = new PCA(training.first().features.size / 2).fit(parsedData.map(_.features))
val training_pca = training.map(p => p.copy(features = pca.transform(p.features)))
val test_pca = test.map(p => p.copy(features = pca.transform(p.features)))

// Run training algorithm to build the model
val numIterations = 200
val stepSize = .000001
val regParam = .001
val model = SVMWithSGD.train(training, numIterations, stepSize, regParam)
val model_pca = SVMWithSGD.train(training_pca, numIterations, stepSize, regParam)

// Clear the default threshold.
model.clearThreshold()
// Clear the default threshold.
model_pca.clearThreshold()

// Compute raw scores on the test set.
val scoreAndLabels = test.map { point =>
  val score = model.predict(point.features)
  (score, point.label)
}

// Compute raw scores on the test set.
val scoreAndLabels_pca = test_pca.map { point =>
  val score = model_pca.predict(point.features)
  (score, point.label)
}

// Get evaluation metrics.
val metrics = new BinaryClassificationMetrics(scoreAndLabels)
val auROC = metrics.areaUnderROC()

// Get evaluation metrics.
val metrics_pca = new BinaryClassificationMetrics(scoreAndLabels_pca)
val auROC_pca = metrics_pca.areaUnderROC()
```

	SVMwithSGD	SVMwithSGD (PCA)
Area under ROC -	0.451490631847	0.4858363297
Summary mean –	[2.546063337, 31.11305277]	[2.546063337, 31.11305277]
Summary variance –	[69.225562376, 20781.010950]	[69.225562376, 20781.010950]
Summary numnonzeroes –	[515345.0, 515345.0]	[515345.0, 515345.0]
Summary max -	[87.913, 3423.59]	[87.913, 3423.59]
Summary min –	[-94.04196, -2025.77816]	[-94.04196, -2025.77816]



```
15/07/11 04:00:43 INFO DAGScheduler: ResultStage 215 (aggregate at AreaUnderCurve.scala:45) finis
15/07/11 04:00:43 INFO TaskSchedulerImpl: Removed TaskSet 215.0, whose tasks have all completed,
15/07/11 04:00:43 INFO DAGScheduler: Job 107 finished: aggregate at AreaUnderCurve.scala:45, took
Area under ROC = 0.45149063184743377
summary.mean = [2.546063357110279,31.113052772880316]
summary.variance = [69.225562376193,20781.010950725195]
summary.numNonzeros = [515345.0,515345.0]
summary.max = [87.91324,3423.59535]
summary.min = [-94.04196,-2025.77816]
15/07/11 04:00:43 INFO SparkContext: Invoking stop() from shutdown hook
15/07/11 04:00:44 INFO SparkUI: Stopped Spark web UI at http://192.168.59.3:4040
15/07/11 04:00:44 INFO DAGScheduler: Stopping DAGScheduler
15/07/11 04:00:44 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
15/07/11 04:00:44 INFO Utils: path = C:\Users\kshitij\AppData\Local\Temp\spark-20e3b2d3-3ddb-4b02-
15/07/11 04:00:44 INFO MemoryStore: MemoryStore cleared
15/07/11 04:00:44 INFO BlockManager: BlockManager stopped
15/07/11 04:00:44 INFO BlockManagerMaster: BlockManagerMaster stopped
```

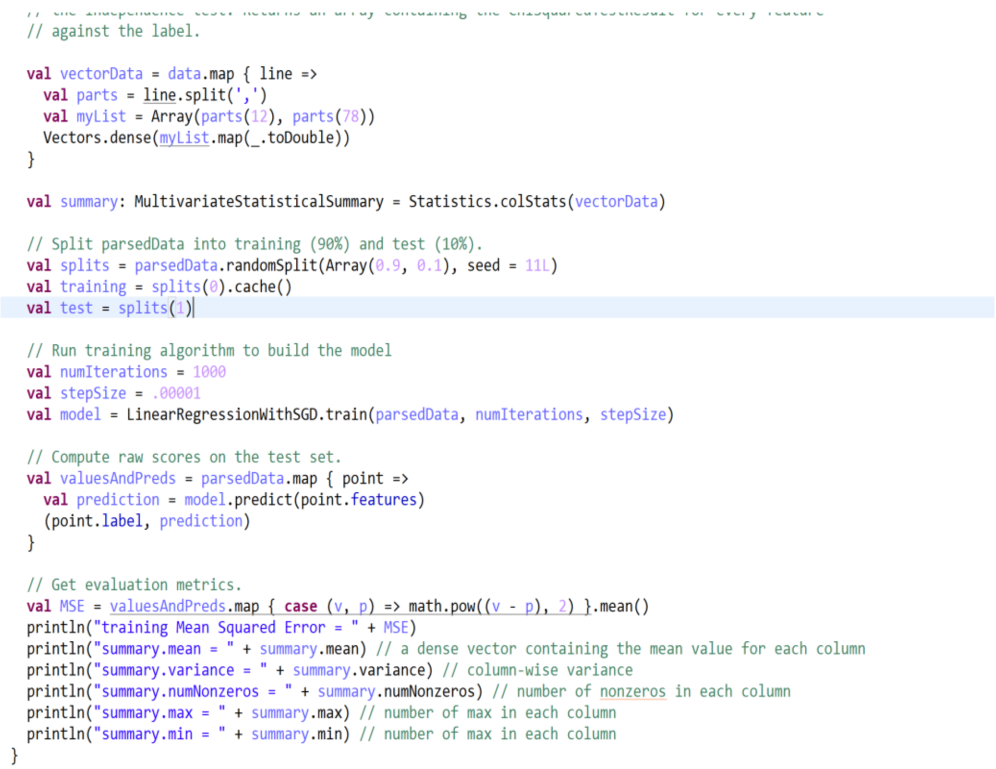
With PCA



```
0.0  0.0  Area under ROC = 0.45149063184743377
Area under ROC with PCA = 0.4858363297193652
summary.mean = [2.546063357110279,31.113052772880316]
summary.variance = [69.225562376193,20781.010950725195]
summary.numNonzeros = [515345.0,515345.0]
summary.max = [87.91324,3423.59535]
summary.min = [-94.04196,-2025.77816]
15/07/11 02:20:20 INFO SparkContext: Invoking stop() from shutdown hook
15/07/11 02:20:20 INFO SparkUI: Stopped Spark web UI at http://192.168.59.3:4040
15/07/11 02:20:20 INFO DAGScheduler: Stopping DAGScheduler
15/07/11 02:20:20 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
15/07/11 02:20:20 INFO Utils: path = C:\Users\kshitij\AppData\Local\Temp\spark-4d672b2b-5d2a-4460-8b38-d6f13ff553d5\blockmgr-35716b12-3b46-4bb5-
15/07/11 02:20:20 INFO MemoryStore: MemoryStore cleared
15/07/11 02:20:20 INFO BlockManager: BlockManager stopped
15/07/11 02:20:20 INFO BlockManagerMaster: BlockManagerMaster stopped
15/07/11 02:20:20 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
15/07/11 02:20:20 INFO SparkContext: Successfully stopped SparkContext
15/07/11 02:20:20 INFO Utils: Shutdown hook called
15/07/11 02:20:20 INFO Utils: Deleting directory C:\Users\kshitij\AppData\Local\Temp\spark-4d672b2b-5d2a-4460-8b38-d6f13ff553d5
15/07/11 02:20:20 INFO RemoteActorRefProvider$RemotingTerminator: Shutting down remote daemon.
15/07/11 02:20:20 INFO RemoteActorRefProvider$RemotingTerminator: Remote daemon shut down; proceeding with flushing remote transports.
15/07/11 02:20:20 INFO RemoteActorRefProvider$RemotingTerminator: Remoting shut down.
```

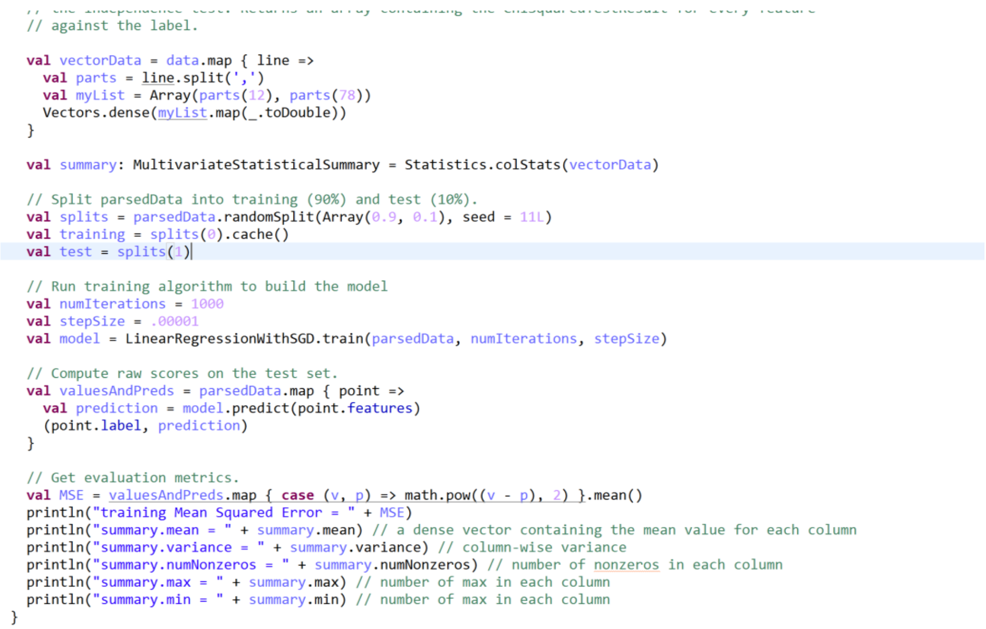
(Regression)

LinearRegressionwithSGD Code Implementation



```
  // the independence test. Returns an array containing the ChiSquaredTestResult for every feature
  // against the label.

  val vectorData = data.map { line =>
    val parts = line.split(',')
    val myList = Array(parts(12), parts(78))
    Vectors.dense(myList.map(_.toDouble))
  }

  val summary: MultivariateStatisticalSummary = Statistics.colStats(vectorData)

  // Split parsedData into training (90%) and test (10%).
  val splits = parsedData.randomSplit(Array(0.9, 0.1), seed = 11L)
  val training = splits(0).cache()
  val test = splits(1)

  // Run training algorithm to build the model
  val numIterations = 1000
  val stepSize = .00001
  val model = LinearRegressionWithSGD.train(parsedData, numIterations, stepSize)

  // Compute raw scores on the test set.
  val valuesAndPreds = parsedData.map { point =>
    val prediction = model.predict(point.features)
    (point.label, prediction)
  }

  // Get evaluation metrics.
  val MSE = valuesAndPreds.map { case (v, p) => math.pow((v - p), 2) }.mean()
  println("training Mean Squared Error = " + MSE)
  println("summary.mean = " + summary.mean) // a dense vector containing the mean value for each column
  println("summary.variance = " + summary.variance) // column-wise variance
  println("summary.numNonzeros = " + summary.numNonzeros) // number of nonzeros in each column
  println("summary.max = " + summary.max) // number of max in each column
  println("summary.min = " + summary.min) // number of max in each column
 }
```

LinearRegressionwithSGD (PCA) Code Implementation



```
  // the independence test. Returns an array containing the ChiSquaredTestResult for every feature
  // against the label.

  val vectorData = data.map { line =>
    val parts = line.split(',')
    val myList = Array(parts(12), parts(78))
    Vectors.dense(myList.map(_.toDouble))
  }

  val summary: MultivariateStatisticalSummary = Statistics.colStats(vectorData)

  // Split parsedData into training (90%) and test (10%).
  val splits = parsedData.randomSplit(Array(0.9, 0.1), seed = 11L)
  val training = splits(0).cache()
  val test = splits(1)

  // Run training algorithm to build the model
  val numIterations = 1000
  val stepSize = .00001
  val model = LinearRegressionWithSGD.train(parsedData, numIterations, stepSize)

  // Compute raw scores on the test set.
  val valuesAndPreds = parsedData.map { point =>
    val prediction = model.predict(point.features)
    (point.label, prediction)
  }

  // Get evaluation metrics.
  val MSE = valuesAndPreds.map { case (v, p) => math.pow((v - p), 2) }.mean()
  println("training Mean Squared Error = " + MSE)
  println("summary.mean = " + summary.mean) // a dense vector containing the mean value for each column
  println("summary.variance = " + summary.variance) // column-wise variance
  println("summary.numNonzeros = " + summary.numNonzeros) // number of nonzeros in each column
  println("summary.max = " + summary.max) // number of max in each column
  println("summary.min = " + summary.min) // number of max in each column
 }
```

LinearRegressionwithSGD LinearRegressionwithSGD (PCA)

Training mean squared error - .75798737993133		.75798737993133
Summary mean –	[2.546063337, 31.11305277]	[2.546063337, 31.11305277]
Summary variance –	[69.225562376, 20781.010950]	[69.225562376, 20781.010950]
Summary numnonzeroes –	[515345.0, 515345.0]	[515345.0, 515345.0]
Summary max -	[87.913, 3423.59]	[87.913, 3423.59]
Summary min –	[-94.04196, -2025.77816]	[-94.04196, -2025.77816]

f)	SVMwithSGD	LinearRegressionSGD
Area under ROC -	0.451490631847	Training mean squared error - .75798737993133

Command for Question 1:

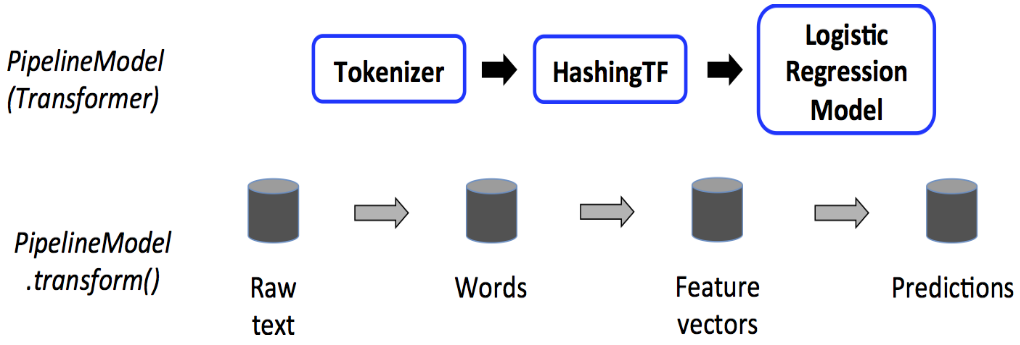
```
C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.midterm.reg.MusicYearPerdictREG" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar
```

```
C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.midterm.reg.MusicYearPerdictREGLASSO" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar
```

```
C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.midterm.reg.MusicYearPerdictREGPCA" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar
```

Question 2—

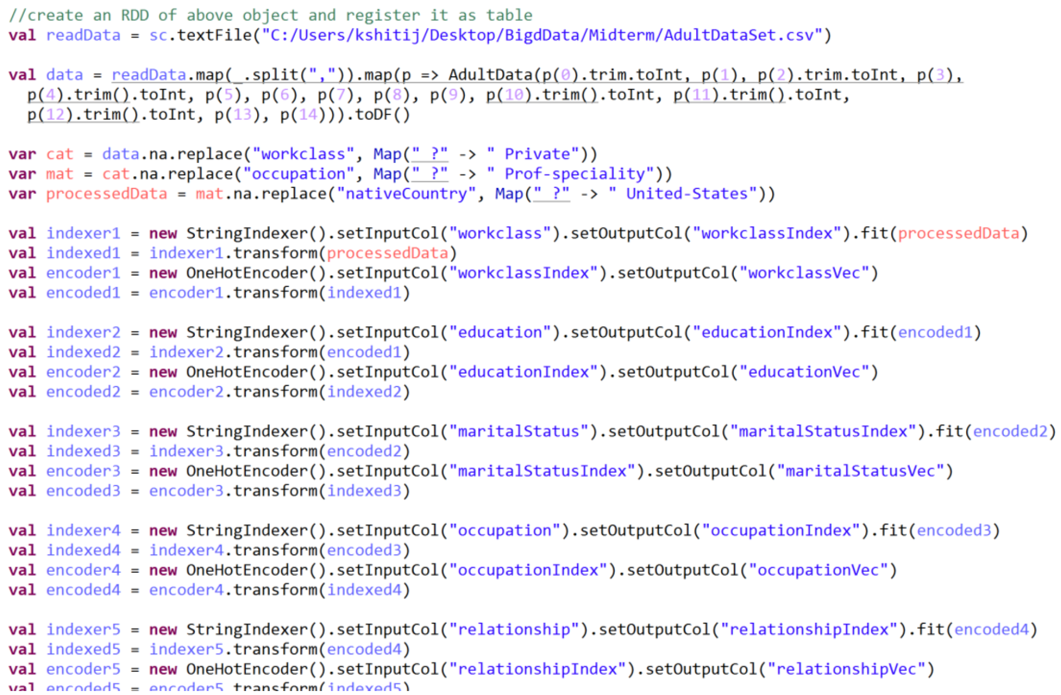
1. Pre-Processing steps –
2. The data present in the file needed to be cleaned as there were some invalid values present like " ?". For that we first created a dataframe using a csv file and then converted the dataframe to dataframeNAFunctions to use the functions to deal with the Na values to clean the data.
3. As the data had categorical values so changing the categorical values to the numerical values was important and for that we used spar
4. OneHotEncoder to convert the categorical dataframes values into numerical ones.
5. Covert the dataframe use the Estimator, Transformer, and ParamMap in order to covert the Pipeline model using Traing dataset frames in Logistic regression Model.
6. Use the Test Data in to predict the result based on the features and labels from the traing data model.



7. Two Different classification methods used were SVMwithSGD and LogisticRegressionwithSGD to build the training set model for the perdiction.

	SVMwithSGD	LogisticRegressionwithSGD
Error	.01135	.01135

Logistic Regression Code



```
//create an RDD of above object and register it as table
val readData = sc.textFile("C:/Users/kshitij/Desktop/BigdData/Midterm/AdultDataSet.csv")

val data = readData.map(_.split(",")).map(p => AdultData(p(0).trim.toInt, p(1), p(2).trim.toInt, p(3),
  p(4).trim().toInt, p(5), p(6), p(7), p(8), p(9), p(10).trim().toInt, p(11).trim().toInt,
  p(12).trim().toInt, p(13), p(14))).toDF()

var cat = data.na.replace("workclass", Map(" ?" -> " Private"))
var mat = cat.na.replace("occupation", Map(" ?" -> " Prof-speciality"))
var processedData = mat.na.replace("nativeCountry", Map(" ?" -> " United-States"))

val indexer1 = new StringIndexer().setInputCol("workclass").setOutputCol("workclassIndex").fit(processedData)
val indexed1 = indexer1.transform(processedData)
val encoder1 = new OneHotEncoder().setInputCol("workclassIndex").setOutputCol("workclassVec")
val encoded1 = encoder1.transform(indexed1)

val indexer2 = new StringIndexer().setInputCol("education").setOutputCol("educationIndex").fit(encoded1)
val indexed2 = indexer2.transform(encoded1)
val encoder2 = new OneHotEncoder().setInputCol("educationIndex").setOutputCol("educationVec")
val encoded2 = encoder2.transform(indexed2)

val indexer3 = new StringIndexer().setInputCol("maritalStatus").setOutputCol("maritalStatusIndex").fit(encoded2)
val indexed3 = indexer3.transform(encoded2)
val encoder3 = new OneHotEncoder().setInputCol("maritalStatusIndex").setOutputCol("maritalStatusVec")
val encoded3 = encoder3.transform(indexed3)

val indexer4 = new StringIndexer().setInputCol("occupation").setOutputCol("occupationIndex").fit(encoded3)
val indexed4 = indexer4.transform(encoded3)
val encoder4 = new OneHotEncoder().setInputCol("occupationIndex").setOutputCol("occupationVec")
val encoded4 = encoder4.transform(indexed4)

val indexer5 = new StringIndexer().setInputCol("relationship").setOutputCol("relationshipIndex").fit(encoded4)
val indexed5 = indexer5.transform(encoded4)
val encoder5 = new OneHotEncoder().setInputCol("relationshipIndex").setOutputCol("relationshipVec")
val encoded5 = encoder5.transform(indexed5)
```

Command for Question 2:

```
C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.midterm.df.AdultDataSet" --
master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar
```

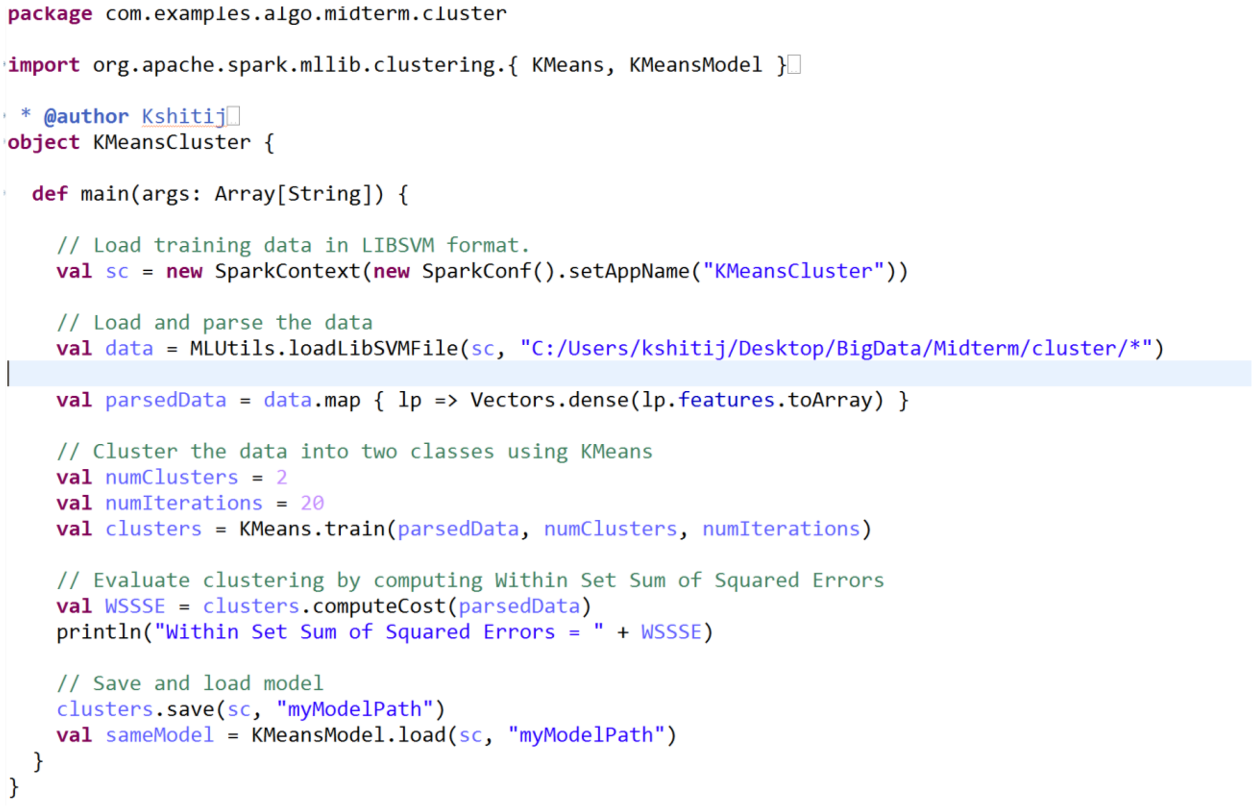
Question 3—

1. Pre-Processing steps used

   For preprocessing in clustering, we saved all the file in one location on the drive from where we can pull all the files at a time using the "*" function so that clustering can be done on those files using the below clustering algorithms

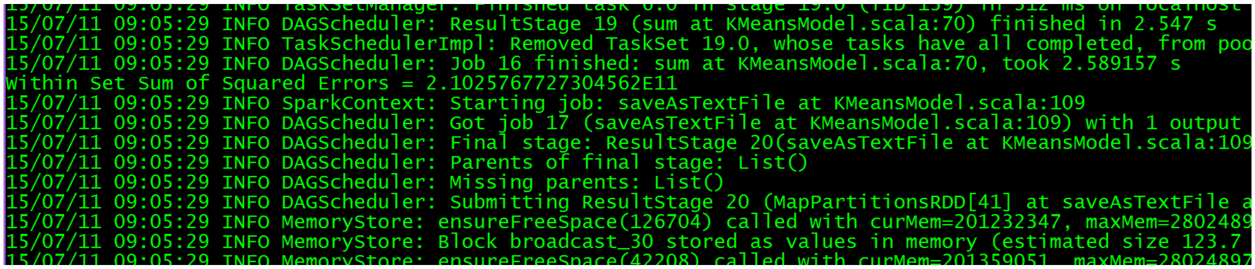
Clustering methods used

a. KMeansCluster Code



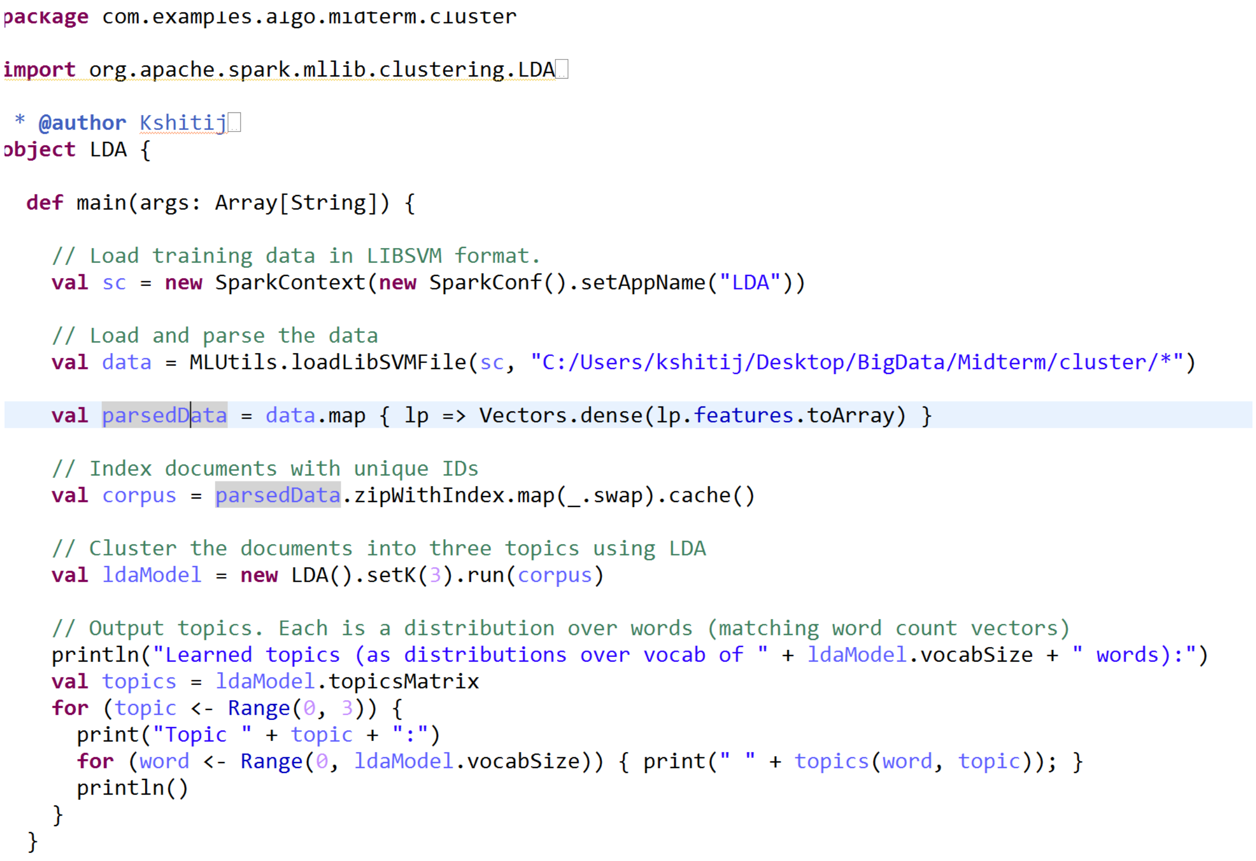
```
package com.examples.algo.midterm.cluster

import org.apache.spark.mllib.clustering.{ KMeans, KMeansModel }

 * @author Kshitij
object KMeansCluster {

  def main(args: Array[String]) {

    // Load training data in LIBSVM format.
    val sc = new SparkContext(new SparkConf().setAppName("KMeansCluster"))

    // Load and parse the data
    val data = MLUtils.loadLibSVMFile(sc, "C:/Users/kshitij/Desktop/BigData/Midterm/cluster/*")

    val parsedData = data.map { lp => Vectors.dense(lp.features.toArray) }

    // Cluster the data into two classes using KMeans
    val numClusters = 2
    val numIterations = 20
    val clusters = KMeans.train(parsedData, numClusters, numIterations)

    // Evaluate clustering by computing Within Set Sum of Squared Errors
    val WSSSE = clusters.computeCost(parsedData)
    println("Within Set Sum of Squared Errors = " + WSSSE)

    // Save and load model
    clusters.save(sc, "myModelPath")
    val sameModel = KMeansModel.load(sc, "myModelPath")
  }
}
```

Within Set Sum of Squared Errors = 2605.254084081905



```
15/07/11 09:05:29 INFO TaskSetManager: Finished task 0.0 in stage 19.0 (TID 139) in 312 ms on localhost
15/07/11 09:05:29 INFO DAGScheduler: ResultStage 19 (sum at KMeansModel.scala:70) finished in 2.547 s
15/07/11 09:05:29 INFO TaskSchedulerImpl: Removed TaskSet 19.0, whose tasks have all completed, from poo
15/07/11 09:05:29 INFO DAGScheduler: Job 16 finished: sum at KMeansModel.scala:70, took 2.589157 s
Within Set Sum of Squared Errors = 2.1025767727304562E11
15/07/11 09:05:29 INFO SparkContext: Starting job: saveAsTextFile at KMeansModel.scala:109
15/07/11 09:05:29 INFO DAGScheduler: Got job 17 (saveAsTextFile at KMeansModel.scala:109) with 1 output
15/07/11 09:05:29 INFO DAGScheduler: Final stage: ResultStage 20(saveAsTextFile at KMeansModel.scala:109
15/07/11 09:05:29 INFO DAGScheduler: Parents of final stage: List()
15/07/11 09:05:29 INFO DAGScheduler: Missing parents: List()
15/07/11 09:05:29 INFO DAGScheduler: Submitting ResultStage 20 (MapPartitionsRDD[41] at saveAsTextFile a
15/07/11 09:05:29 INFO MemoryStore: ensureFreeSpace(126704) called with curMem=201232347, maxMem=2802489
15/07/11 09:05:29 INFO MemoryStore: Block broadcast_30 stored as values in memory (estimated size 123.7
15/07/11 09:05:29 INFO MemoryStore: ensureFreeSpace(42208) called with curMem=201359051, maxMem=28024897
```

b. LDA – Latent Dirichlet allocation Code Implementation



```
package com.examples.algo.midterm.cluster

import org.apache.spark.mllib.clustering.LDA

 * @author Kshitij
object LDA {

  def main(args: Array[String]) {

    // Load training data in LIBSVM format.
    val sc = new SparkContext(new SparkConf().setAppName("LDA"))

    // Load and parse the data
    val data = MLUtils.loadLibSVMFile(sc, "C:/Users/kshitij/Desktop/BigData/Midterm/cluster/*")

    val parsedData = data.map { lp => Vectors.dense(lp.features.toArray) }

    // Index documents with unique IDs
    val corpus = parsedData.zipWithIndex.map(_.swap).cache()

    // Cluster the documents into three topics using LDA
    val ldaModel = new LDA().setK(3).run(corpus)

    // Output topics. Each is a distribution over words (matching word count vectors)
    println("Learned topics (as distributions over vocab of " + ldaModel.vocabSize + " words):")
    val topics = ldaModel.topicsMatrix
    for (topic <- Range(0, 3)) {
      print("Topic " + topic + ":")
      for (word <- Range(0, ldaModel.vocabSize)) { print(" " + topics(word, topic)); }
      println()
    }
  }
```

Command for Question 3:

```
C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class
"com.examples.algo.midterm.cluster.KMeansCluster" --master local[1] target/spark-scala-maven-
project-0.0.1-SNAPSHOT.jar
```

```
C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class
"com.examples.algo.midterm.cluster.GausianMixCluster" --master local[1] target/spark-scala-maven-
project-0.0.1-SNAPSHOT.jar
```

```
C:/spark-1.4.0-bin-hadoop2.6/bin/spark-submit --class "com.examples.algo.midterm.cluster.LDA" --master local[1] target/spark-scala-maven-project-0.0.1-SNAPSHOT.jar
```