Project Report

TEAM MEMBERS

1. Sai Bhargavi Velugoti
2. Nikhila Raya
3. Sree Siva Sandeep Palaparthi

PROJECT OVERVIEW

In this project, we are focusing on building fraud detection models to predict if a particular transaction is could be a fraudulent using the real-world payment data. The dataset we used for training and testing the models contains information regarding various transactions and attributes related to these transaction.The dataset has interesting attributes such as, isFraud, isFlaggedFraud which we used for classification and predictions.

Using Spark MLlib, we built two ensembled models, a random forest model of decision trees, gradient boosted tree model and also the decision tree model to classify and predict if a particular transaction is fraud. Also in Map-Reduce, we implemented the nearest-neighbor classification algorithm to predict if the transaction is fraudulent. As an enhancement, the nearest-neighbor algorithm was implemented using top-k style pruning.

INPUT DATA

Fraudulent Transactions

https://www.kaggle.com/ntnu-testimon/paysim1

The dataset contains information regarding various transactions and attributes related to these transactions like, the type of transactions (Cash-In, Cash-Out, Debit, Payment or transfer), the details of the user who initiated the transaction and the recipient information. The dataset also includes attributes, such as

isFraud: This attribute describes the fraudulent behavior of the agents by taking control of customers accounts and trying to empty the funds by transferring to another account and then cashing out of the system.

isFlaggedFraud: This attribute aims to control massive transfers from one account to another and flags illegal attempts.

Sample line of the dataset

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
1	PAYMENT	9839.64	C1231006815	170136	160296.36	M1979787155	0	0	0	0

TASKS

Task 1: Classification and prediction in Spark MLlib

***Overview*:**

Using Spark MLlib, we built two ensembled models, a random forest model of decision trees, gradient boosted tree model and also the decision tree model to classify and predict if a particular transaction is fraud. In this task, we have cleaned and formatted each record of the dataset to a transaction schema

record. The data has been split, such that the training data contains 70% of the records and the rest is considered as testing data. Both these splits include fraud transactions and non-fraudulent transactions. The trained data is used to build models and based on these models, predictions are made to each transaction record in the test data. Accuracy is calculated based on the comparison between the actual label of the test record and the predicted label.

Pseudocode:

```
// read input file and convert all values to double and create transaction objects
   var inputDF = parseNumber(sc.textFile(args(0)))
                        .map(inputline => Transaction(inputline(0), inputline(1), inputline(2), inputline(3),
                                          inputline(4), inputline(5),inputline(6), inputline(7),inputline(8),
                                          inputline(9), inputline(10))).toDF().cache()

   /* Forming the features vector by selecting the feature columns. We have considered
    * feature columns as all the columns other than the isFraud column */

   val featureColumns = Array("step","ttype","amt","nameOrig",
                         "oldBalOrg","newBalOrg","nameDest",
                         "oldBalDest","newBalDest","isFlagged")

   val fea_classifier = new VectorAssembler().setInputCols(featureColumns).setOutputCol("features")

   /* Forms a features data frame with a new column features which is an array of all the values of
    * the featureColumns selected above*/
     val featuresDF = fea_classifier.transform(inputDF)

   // Defining the isFraud column as the label by using a StringIndexer
   val label_index = new StringIndexer().setInputCol("isFraud").setOutputCol("label")

   // Forming data frame with a new column label added to the featuresDF dataframe
   val labelDF = label_index.fit(featuresDF).transform(featuresDF)

   /* Splitting the data into train and test data such that fraud transaction records and
    * non fraud transaction records are present in both test and train data */

   val inputFraud = labelDF.filter(x => x.getDouble(9) == 1.0).randomSplit(Array(0.7, 0.3), seed = 20L)
   val inputNotFraud = labelDF.filter(x => x.getDouble(9) == 0.0).randomSplit(Array(0.7, 0.3), seed =
20L)

   var train = inputFraud(0).union(inputNotFraud(0)).toDF()
   var test = inputFraud(1).union(inputNotFraud(1)).toDF()

   // Creating the Random forest classifier, by setting the parameters for training
```

```
val fraudClassifier = new RandomForestClassifier().setImpurity("gini").setMaxDepth(10)
                    .setNumTrees(10).setFeatureSubsetStrategy("auto").setSeed(3000)

// Using the classifier to train the model
val trainedModel = fraudClassifier.fit(train)

// Printing the random forest trees
trainedModel.toDebugString

// Using the trained model on test data to get predictions
val fraudPrediction = trainedModel.transform(test)

// Evaluating the accuracy of the predictions using the binary classification evaluator
val evaluator = new BinaryClassificationEvaluator().setLabelCol("label")
val accuracy = evaluator.evaluate(fraudPrediction)

logger.info("Random Forest accuracy : "+accuracy)

/* Boosted Strategy
 * Converting the train and test into an RDD of LabeledPoints
 */
val trainBoosted = train.rdd.map(trans =>
  new LabeledPoint(trans.getAs("label"),
 org.apache.spark.mllib.linalg.Vectors.fromML(trans.getAs("features"))))
val testBoosted = test.rdd.map(trans =>
  new LabeledPoint(trans.getAs("label"),
    org.apache.spark.mllib.linalg.Vectors.fromML(trans.getAs("features"))))

// Initializing the strategy and setting parameters
val boostingStrategy = BoostingStrategy.defaultParams("Classification")

boostingStrategy.numIterations = 10
boostingStrategy.treeStrategy.numClasses = 2
boostingStrategy.treeStrategy.maxDepth = 10
boostingStrategy.treeStrategy.categoricalFeaturesInfo = Map[Int, Int]()

// Training the model using the train data and the strategy
val boostedModel = GradientBoostedTrees.train(trainBoosted, boostingStrategy)

// Using the trained model on test data to get predictions
val testErrBoosted = testBoosted.map{ point =>
  val prediction = boostedModel.predict(point.features)
  (point.label, prediction)
```

```
    }

    /* Evaluating the accuracy of the predictions by comparing the predicted label with
            the actual label of the test record */
    val testErr = testErrBoosted.filter(r => r._1 == r._2).count.toDouble / testBoosted.count()
    logger.info("Boosted Strategy accuracy : "+testErr)

    /* Decision Tree
     * Defining the train classifier and setting parameters  */
    val numClasses = 2
    val categoricalFeatures = Map[Int,Int]()
    val impurity = "entropy"
    val maxDepth = 10
    val maxBins = 30

    // Training the model using the train data and the parameters
    val model = DecisionTree.trainClassifier(trainBoosted, numClasses, categoricalFeatures, impurity,
maxDepth, maxBins)

    // Using the trained model on test data to get predictions
    var labelprediction = testBoosted.map{ testRecord => val prediction model.predict(testRecord.features)
(testRecord.label, prediction)}

    /* Evaluating the accuracy of the predictions by comparing the predicted label with
    the actual label of the test record */
    val accuracyDecision = labelprediction.filter(row => row._1 == row._2).count().toDouble / test.count()
    logger.info("Decision Tree Accuracy  : "+accuracyDecision);
    logger.info("Decision Tree Model Trained: "+model.toDebugString);

    // Saving the accuracy determined from all the three models to an RDD to save it to an output file
    val finalAccuracyRDD = sc.parallelize(List(Row("Random Forest accuracy",accuracy),
                                    Row("Boosted Strategy accuracy",testErr),
                                    Row("Decision Tree Accuracy",accuracyDecision)))

    finalAccuracyRDD.saveAsTextFile(args(1))
  }

  /* Method to convert the RDD of Strings to RDD of Double
   * The fields nameOrig and nameDest have one character ('C' or 'M')
   * at the start of the value. This character is dropped to convert
   * the field to double.
   * The field type which contains one of 'PAYMENT','TRANSFER','DEBIT'
   * ,'CASH_IN','CASH_OUT'. Inorder to convert the data to double we have
```

```
  * assigned data with the following values
  * PAYMENT - 4.0, TRANSFER - 5.0, DEBIT - 3.0, CASH_IN - 1.0 and CASH_OUT - 2.0  */

 def parseNumber(inputRDD : RDD[String]): RDD[Array[Double]] = {
  val parsedArray = inputRDD.map(_.split(","))
                    .map{case
Array(step,ptype,amt,nameOrig,oldBalOrg,newBalOrg,nameDest,oldBalDest,newBalDest,isFraud,isFlag
ged)
                                    => return parsedArray; }

 // Defining transaction schema for each record
 case class Transaction( step: Double, ttype: Double, amt: Double, nameOrig: Double,
                         oldBalOrg: Double, newBalOrg: Double, nameDest: Double,
                         oldBalDest: Double,newBalDest: Double, isFraud: Double, isFlagged: Double)}
```

Algorithm and Program Analysis

	Depth	Trees	Bins/Iterations	Time	Accuracy
Random Forest	5	2	NA	4 min	0.949547762376862
Gradient Boosted	5	NA	3	4 min	0.9995864554204178
Decision Tree	5	NA	10	4 min	0.9994754789003274
Random Forest	10	2	NA	8 min	0.9948214461720489
Gradient Boosted	10	NA	5	8 min	0.9996398497838576
Decision Tree	10	NA	10	8 min	0.9995335845311294
Random Forest	15	500	32	2 hrs 38 min	0.9948214461720489

Experiments

6 m4.large machines (1 master and 5 workers)
11 m4.large machines (1 master and 10 workers)

Speedup

Number of machines	Running Time
11 machines	1 hr 13 min

6 machines	1 hr 48 min

Increasing the number of trees, the depth or the number of iterations increases the running time on AWS.

Scalability

Data Size	Running Time
6M records	1 hr 13 min
3M records	38 min

Result Sample
[Random Forest accuracy, 0.9948214461720489]
[Boosted Strategy accuracy, 0.9996398497838576]
[Decision Tree Accuracy, 0.9995335845311294]

Task 2: Nearest neighbor classifier - Complete Data Transfer to Reducers after finding distance
Overview:
Implemented the nearest neighbor algorithm to predict if the transaction is fraudulent or not. The distance is calculated based on three attributes, that is, the amount, transaction type(CREDIT, DEBIT etc) and the user ID of the customer performing the transaction. Distance from each test record to each train record is set from Mappers to Reducers.

Pseudocode:

```
Mapper{
        setup ()  {
        Load test data file from cache and store in test list
        k = retrieve k value from context
        }
        // each record from training data
        map() {
        For each test record in test list
                measure the distance to the training record mapped to this call
                // distance measure
                For string attributes, distance is 0 if the strings are equal, else distance is 1
                For numerical attributes, distance is the difference between the values ( Will modify the
                calculation using normalized values)
                Write the (testKey, (distance, classLabel)) with key for the testrecord in increasing order
                of distance.
        }
```

```
}

Reducer {
        setup(){ k = retrieve k value from context }
        reduce (testKey, values){
                // values contain distance and classLabel to the training records.
                For each (dist, label) in values
                        add (dist,label) to map with distance as key, maintaining k nodes in the map.
                        // call it distanceMap
                Classlabels = values from the distanceMap
                For each classlabel from Classlabels
                        Count the frequency in distanceMap
                Output the label that has highest frequency
        }
}
```

Experiments:

AWS cluster size and machine type : 11 m4.large machines (1 master and 10 workers)
For test data size of 100Mb and train data size of 500Mb, program is running for >3hr without terminating.
For a small test set and large test set, using Broadcast join, most of the records are predicted as false negative, as the train data is non-uniformly distributed - records with more negative class labels and less positive class labels

Result Sample:

TestRecord	ClassLabel
167715.95,C1154341183,1.346656443E7,1.363428038E7,C918598842,490266.1,356684.37,0,0	0

Task 3: Nearest neighbor classifier - Complete Data Transfer to Reducers after finding distance

Overview:

Implemented the nearest neighbor algorithm to predict if the transaction is fraudulent or not. The distance is calculated based on three attributes, that is, the amount, transaction type(CREDIT, DEBIT etc) and the user ID of the customer performing the transaction. This is an improvisation to Task 2, where the data transferred from Mappers to Reducers is cut down using Top-K pattern. We used HashMap and TreeMap to achieve this.

```
Mapper{
HashMap to store the test record ID and the corresponding TreeMap with distance and class labels of the nearest k train records
HashMap<String, TreeMap<Double, String>> localTopK = new HashMap<String, TreeMap<Double, String>>();

        setup () {
        Load test data file from cache and store in test list
```

```
        k = retrieve k value from context
        }
        // each record from training data
        map() {
        For each test record in test list
                measure the distance to the training record mapped to this call
                // distance measure
                For string attributes, distance is 0 if the strings are equal, else distance is 1
                For numerical attributes, distance is the difference between the values ( Will modify the
                calculation using normalized values)
                For this test record in the localTopK, insert the (distance, classLabel) into the
                corresponding TreeMap
                if(localTopK.get(testrecord).size()) > k)
                        Remove the entry with greater distance
        }
        cleanup(){
        For each entry in localTopK
                emit (testKey, (distance, classLabel))
        }
}
Reducer {
        setup(){ k = retrieve k value from context }
        reduce (testKey, values){
                // values contain distance and classLabel to the training records.
                For each (dist, label) in values
                        add (dist,label) to map with distance as key, maintaining k nodes in the map.
                        // call it distanceMap
                Classlabels = values from the distanceMap
                For each classlabel from Classlabels
                        Count the frequency in distanceMap
                Output the label that has highest frequency
        }
}
```

Experiments:

AWS cluster size and machine type : 11 m4.large machines (1 master and 10 workers)

For large test and train data, program is running for longer time without terminating.

For a small test set and large test set, using Broadcast join, most of the records are predicted as false negative, as the train data is non-uniformly distributed - records with more negative class labels and less positive class labels

Result Sample:

TestRecord	ClassLabel
167715.95,C1154341183,1.346656443E7,1.363428038E7,C918598842,490266.1,356684.37,0,0	0

Metrics: Accuracy - 88%

Conclusion

We were able to predict the fraudulent transactions using the models classified by k-nearest neighbor, decision tree and ensemble methods - Random Forest, Gradient Boosted trees. The KNN algorithm implemented can be improved using block partitioning(1-Bucket) to split and duplicate both test and train records by including a new job to assign a region ID to each test and train record. Distance measure can be improved by considering several other attributes. Also in the above tasks, secondary sort can be used to reduce the overhead of sorting in the reduce tasks.

References:

http://ijssst.info/Vol-15/No-3/data/3857a513.pdf
https://spark.apache.org/docs/latest/mllib-ensembles.html

CCIS Github repository: https://github.ccs.neu.edu/cs6240f18/Worthless-Without-Coffee/tree/master