Assignment #3

COMP 424 Big Data

Qiangqiang Li (liqiangq@gmail.com)

2 Individual project

(a). (15 marks) Describe the task including the details of the input data (data source, data size, the original number of features and instances, feature types, whether missing values exist, etc.) and the expected output of the system;

Answer:

- The task is using Decision Tree algorithm to deal with text classification, this is because Decision Trees are a class of very powerful Machine Learning model cable of achieving high accuracy in many tasks while being highly interpretable.
- The input data is Sensorless Drive Diagnosis Data Set, which the detail like the below table, includes features(48) and instances(58509), feature types(Real), whether missing values exist(N/A).

Data source	https://archive.ics.uci.edu/ml/datasets/dataset+for+sensorless+drive+diagnosis#				
Data Set Characteristics:	Multivariate	**Number of Instances:**	58509	**Area:**	Computer
Attribute Characteristics:	Real	**Number of Attributes:**	49	**Date Donated**	2015-02-24
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	56382

- Features are extracted from electric current drive signals. The drive has intact and defective components. This results in 11 different classes with different conditions. Each condition has been measured several times by 12 different operating conditions, this means by different speeds, load moments and load forces. The current signals are measured with a current probe and an oscilloscope on two phases.
- The expected output of the system is training and testing the dataset, throng these 48 features to Predict the different classes which the motor has intact and defective components.

(b) (5 marks) Describe all the preprocessing steps that are applied to the download data file(s) to obtain the dataset that is used as input of your program, which must be at least 20MB and have at least 30 features.

Answer:

- Sensorless Drive Diagnosis Data Set is from the website, and the original data is about 25M, and using the format of file is text. So need to transform the format to the csv file.
- Because the Features of dataset are number, the good way is using spark mllib .classification lib to transform the JavaRDD, the preprocessing data is format the dataset, the string style translate to double.

- The dataset file (Sensorless_drive_diagnosis.csv) include 58509, each data has 48 Features and one label class, and the file size is about 24M .
- **Normalizing data:** use weka tool to preprocessing data, Sensorless_drive_diagnosis.csv Transform to Sensorless_drive_diagnosis_normalising.csv. the file size become a smaller one , about 23M.
- **Using PCA:** use weka tool to preprocessing data, Sensorless_drive_diagnosis.csv Transform to Sensorless_drive_diagnosis_pca.csv. the file size become a smaller one , about 11M,and feather

(c) (15 marks) Describe the program using UML class diagrams and/or pseudo-code;

Answer: The program using the spark.ml lib and Decision Tree model:

1. **The initialization of Spark Session, load the dataset and convert TO Java RDD.**

```
SparkSession spark = SparkSession.builder().appName("Decision_Tree").
                                        //          master("local").
                                                    getOrCreate();

Dataset<Row> kddCSV = spark.read().format("csv").load(data_path);
kddCSV.show();

// convertTOJavaRDD
JavaRDD<String> lines = spark.sparkContext().textFile(data_path, 0).toJavaRDD();
JavaRDD<LabeledPoint> linesRDD = lines.map(line -> {
                                        String[] tokens = line.split(",");
                                        double[] features = new double[tokens.length - 1];
                                        for (int i = 0; i < features.length; i++) {
                                                    features[i] = Double.parseDouble(tokens[i]);
                                        }
                                        DenseVector v = new DenseVector(features);
                                        double labelname = Double.parseDouble(tokens[features.length]);
                                        return new LabeledPoint(labelname, v);
});
```

2. **Index labels, adding meta data to the label column. Fit on whole data set to include all labels in index.**

```
// create the data frame
Dataset<Row> data = spark.createDataFrame(linesRDD, LabeledPoint.class);
data.show();
StringIndexerModel labelIndexer = new StringIndexer().setInputCol("label").setOutputCol("indexedLabel")
                                                      .fit(data);
```

3. **Automatically identify categorical features, and index them, features with > 4 distinct values are treated as continuous.**

```
// Automatically identify categorical features, and index them.
VectorIndexerModel featureIndexer = new
            VectorIndexer().setInputCol("features").setOutputCol("indexedFeatures")
             .setMaxCategories(4) // features with > 4 distinct values are treated as continuous
                        .fit(data);
```

4. Split the data into training and test sets (30% held out for testing), Split the data into t Train a Decision Tree model, and the initialization of DecisionTreeClassifier.

```
// Split the data into training and test sets (30% held out for testing)
Dataset<Row>[] splits = data.randomSplit(new double[] { 0.7, 0.3 });
Dataset<Row> training_set = splits[0];
Dataset<Row> test_set = splits[1];

// Train a DecisionTree model
DecisionTreeClassifier dt = new DecisionTreeClassifier().setLabelCol("indexedLabel")
                                        .setFeaturesCol("indexedFeatures");
```

5. Chain indexers and tree in a Pipeline, and using PipelineModel on training data, Make predictions, Select example rows to display.

```
// Convert indexed labels back to original labels.
                    IndexToString labelConverter = new
IndexToString().setInputCol("prediction").setOutputCol("predictedLabel")
                                        .setLabels(labelIndexer.labels());

                    // Chain indexers and tree in a Pipeline
                    Pipeline pipeline = new Pipeline()
                                        .setStages(new PipelineStage[] { labelIndexer, featureIndexer, dt,
labelConverter });
                    // Train model. This also runs the indexers.
                    PipelineModel model = pipeline.fit(training_set);

                    // Make predictions.
                    Dataset<Row> train_predictions = model.transform(training_set);
                    Dataset<Row> test_predictions = model.transform(test_set);

                    // Select example rows to display.
                    test_predictions.select("predictedLabel", "label", "features").show();

                    // Select (prediction, true label) and compute test error
                    MulticlassClassificationEvaluator evaluator = new MulticlassClassificationEvaluator

          .setLabelCol("indexedLabel").setPredictionCol("prediction").setMetricName("accuracy");
```

6. Print the result, include test accuracy, training accuracy, the running time , and the model.

```
double test_accuracy = evaluator.evaluate(test_predictions);
double training_accuracy = evaluator.evaluate(train_predictions);
System.out.println("test error: " + (1.0 - test_accuracy));
System.out.println("test accuracy: " + test_accuracy);
System.out.println("train accuracy: " + training_accuracy);
DecisionTreeClassificationModel treeModel = (DecisionTreeClassificationModel) (model.stages()[2]);
System.out.println("desision tree model:\n" + treeModel.toDebugString());
long endTime = System.currentTimeMillis();
float running_time = (endTime - startTime) / 1000;
System.out.println("running time： " + running_time + "ms");
```

(d) (5 marks) Describe how to install and run the program step by step in a Readme.txt file.

Answer: The steps for applying Decision Tree. The contents of Readme.txt is listed below:

Step 1: upload java program: Individual_DecisionTree.java

Step 2: ssh to co246a-1.ecs.vuw.ac.nz, and then run environment variables. It was edited a HadoopSetup.csh to avoid inputting the commands manually each time. The contents of HadoopSetup.csh is:

- export HADOOP_VERSION=2.8.0
- export HADOOP_PREFIX=/local/Hadoop/hadoop-$HADOOP_VERSION
- export PATH=${PATH}:$HADOOP_PREFIX/bin
- export SPARK_HOME=/local/spark/spark-2.4.2-bin-hadoop2.7
- export PATH=${PATH}:$HADOOP_PREFIX/bin:$SPARK_HOME/bin
- export HADOOP_CONF_DIR=$HADOOP_PREFIX/etc/hadoop
- export YARN_CONF_DIR=$HADOOP_PREFIX/etc/Hadoop
- export LD_LIBRARY_PATH=$HADOOP_PREFIX/lib/native:$JAVA_HOME/jre/lib/amd64/server

The spark-hadoop package is already existed in the /local/spark/spark-2.4.2-hadoop2.7 in clusters;

- **Step 3**: upload the dataset: Sensorless_drive_diagnosis.csv to one of the Hadoop clusters under /user/<USERNAME>/input. In this case, co246a-1.ecs.vuw.ac.nz is the cluster we used. The command to transfer Sensorless_drive_diagnosis.csv is: $hdfs dfs -put Sensorless_drive_diagnosis.csv /user/<USERNAME>/input;
- **Step 4**: copy all the libs from /local/spark/spark-2.4.2-hadoop2.7/jars to the working folder;
- **Step 5**: Go to the working folder, make two new class folder: idt_class by command: #mkdir idt_class;
- **Step 6**: patch the java programs to jar patches:
  #javac -cp “libs/*” -d idt_class Individual_DecisionTree.java
  #jar cvf Individual_DecisionTree.jar idt_class/ .
- **Step 7**: Submit the work to spark Hadoop jobs:
  #spark-submit --class "nz.ac.vuw.ecs.Individual_DecisionTree" --master yarn --deploy-mode cluster Individual_DecisionTree.jar /user/<USERNAME>/input/Sensorless_drive_diagnosis.csv.

(e) (15 marks) Compare and discuss the results (including the training and test accuracy, the running time, the model, etc. depending on the program) of the program with and without normalising/scaling data.

Answer:

	Training Acc	Test Acc	Running time	The model	program
Original data	72.84%	73.08%	36ms	DecisionTreeClassificationModel Depth 5 with 39 nodes	nz.ac.vuw.ecs.Individual_DecisionTree.java
normalizing data	81.11%	80.46%	32ms	DecisionTreeClassificationModel Depth 5 with 47 nodes	nz.ac.vuw.ecs.Individual_DecisionTree_N.java

Observably, the scaling data using the same program has a more accurate result than the original data about nearly 9% higher in both training and test dataset. Because using data normalization in order to seek for relation in features, and the normalization data will be better to training and test these features, and give the higher accurate.

On the other hand, the two data sets have the same size, so the running time are the similarly, and use the same model (DecisionTreeClassificationModel), but the result in model has the slight difference, this is may be the seed random.

In summary, the normalization data will be better than the original data to use the Decision Tree algorithm deal with text classification in this case.

(f) (15 marks) Compare and discuss the results (including the training and test accuracy, the running time, the model, etc. depending on the program) of the program with and without transforming data using PCA.

	Training Acc	Test Acc	Running time	Data information	The model	program
Original data	72.84%	73.08%	36ms	Size:24M Features:48	Depth 5 with 39 nodes	nz.ac.vuw.ecs.Individual_DecisionTree.java
PCA	52.62%	51.97%	33ms	Size:11M Features:21	Depth 5 with 53 nodes	nz.ac.vuw.ecs.Individual_DecisionTree_PCA.java

Observably, with transforming data using PCA data has a less accurate result than the original data about nearly 20% lower in both training and test dataset. It is mean that using PCA will not suit for this dataset, because PCA is sensitive to the relative scaling of the original variables, the data size become smaller, but the accurate become lower is the mean that these features are relation with each other , the features become less, the means will lack the core mean. It is clear that the transforming data using PCA of the Decision Tree Classification method has not a good way to using in this data.

More over, the running times are the same, but the two data set size have the huge different, and the model (DecisionTreeClassificationModel) using the two ways have the same running time, and the model show the little different in the number of nodes. This is change by the seed random.

In summary, with transforming data using PCA data is worse than the original data to use the Decision Tree algorithm deal with text classification.

Appendix A. Decision Tree Model for Dataset

http://co246a-9.ecs.vuw.ac.nz:8088/proxy/application_1558496412184_1014/ **Original**
http://co246a-9.ecs.vuw.ac.nz:8088/proxy/application_1558496412184_1026/ **normalising**
http://co246a-9.ecs.vuw.ac.nz:8088/proxy/application_1558496412184_1021/ **PCA**

```
test error: 0.26917839310031777
test accuracy: 0.7308216068996822
train accuracy: 0.7284823284823285
desision tree model:
DecisionTreeClassificationModel (uid=dtc_4805073dfeeb) of depth 5 with 39 nodes
  If (feature 9 <= -0.05169999999999996)
   If (feature 6 <= -0.0303405)
    If (feature 33 <= 0.0142755)
     Predict: 5.0
    Else (feature 33 > 0.0142755)
     Predict: 4.0
   Else (feature 6 > -0.0303405)
    If (feature 6 <= 0.021951)
     If (feature 6 <= -0.00289445)
      Predict: 4.0
     Else (feature 6 > -0.00289445)
      Predict: 8.0
    Else (feature 6 > 0.021951)
     If (feature 30 <= -0.0141175)
      If (feature 12 <= 0.0022204499999999997)
       Predict: 8.0
      Else (feature 12 > 0.0022204499999999997)
       Predict: 0.0
     Else (feature 30 > -0.0141175)
      Predict: 7.0
  Else (feature 9 > -0.05169999999999996)
   If (feature 10 <= 0.066537)
    If (feature 10 <= -0.0103795)
     If (feature 8 <= -0.0029812)
      If (feature 11 <= -0.021779)
       Predict: 4.0
      Else (feature 11 > -0.021779)
       Predict: 3.0
     Else (feature 8 > -0.0029812)
      If (feature 7 <= 0.0203365)
       Predict: 8.0
      Else (feature 7 > 0.0203365)
       Predict: 7.0
    Else (feature 10 > -0.0103795)
     If (feature 10 <= 0.031463500000000005)
      If (feature 8 <= 0.016307500000000003)
       Predict: 1.0
      Else (feature 8 > 0.016307500000000003)
       Predict: 2.0
     Else (feature 10 > 0.031463500000000005)
      Predict: 6.0
   Else (feature 10 > 0.066537)
    If (feature 6 <= 0.020324500000000002)
     If (feature 6 <= 0.0025348000000000002)
      If (feature 0 <= -1.995E-5)
       Predict: 4.0
      Else (feature 0 > -1.995E-5)
       Predict: 3.0
     Else (feature 6 > 0.0025348000000000002)
      Predict: 6.0
    Else (feature 6 > 0.020324500000000002)
     If (feature 27 <= 0.022981)
      Predict: 9.0
     Else (feature 27 > 0.022981)
      If (feature 4 <= -1.145E-5)
       Predict: 6.0
      Else (feature 4 > -1.145E-5)
       Predict: 9.0
```

running time：36.0ms

```
test error: 0.19532449803765428
test accuracy: 0.8046755019623457
train accuracy: 0.8111317435496481
desision tree model:
DecisionTreeClassificationModel (uid=dtc_bd954f952a59) of depth 5 with 47 nodes
```

```
If (feature 11 <= 0.4954545)
 If (feature 10 <= 0.36408050000000003)
  If (feature 7 <= 0.650576)
   If (feature 9 <= 0.2970755)
    If (feature 6 <= 0.5088895)
     Predict: 5.0
    Else (feature 6 > 0.5088895)
     Predict: 4.0
   Else (feature 9 > 0.2970755)
    If (feature 11 <= 0.34410850000000004)
     Predict: 4.0
    Else (feature 11 > 0.34410850000000004)
     Predict: 3.0
  Else (feature 7 > 0.650576)
   If (feature 6 <= 0.7656795000000001)
    If (feature 24 <= 0.3585205)
     Predict: 0.0
    Else (feature 24 > 0.3585205)
     Predict: 8.0
   Else (feature 6 > 0.7656795000000001)
    If (feature 10 <= 0.3408775)
     Predict: 7.0
    Else (feature 10 > 0.3408775)
     Predict: 0.0
 Else (feature 10 > 0.36408050000000003)
  If (feature 11 <= 0.4348355)
   If (feature 8 <= 0.7336754999999999)
    If (feature 6 <= 0.5769925)
     Predict: 3.0
    Else (feature 6 > 0.5769925)
     Predict: 1.0
   Else (feature 8 > 0.7336754999999999)
    If (feature 8 <= 0.8016485)
     Predict: 10.0
    Else (feature 8 > 0.8016485)
     Predict: 2.0
  Else (feature 11 > 0.4348355)
   If (feature 11 <= 0.45491550000000003)
    If (feature 33 <= 0.6462060000000001)
```

```
        Predict: 6.0
       Else (feature 33 > 0.6462060000000001)
        Predict: 10.0
      Else (feature 11 > 0.45491550000000003)
       Predict: 6.0
   Else (feature 11 > 0.4954545)
    If (feature 6 <= 0.7724614999999999)
     If (feature 18 <= 0.33653500000000003)
      Predict: 9.0
     Else (feature 18 > 0.33653500000000003)
      If (feature 6 <= 0.657035)
       If (feature 0 <= 0.702442)
        Predict: 4.0
       Else (feature 0 > 0.702442)
        Predict: 3.0
      Else (feature 6 > 0.657035)
       Predict: 6.0
    Else (feature 6 > 0.7724614999999999)
     If (feature 27 <= 0.3949355)
      If (feature 6 <= 0.7879290000000001)
       If (feature 30 <= 0.3612145)
        Predict: 9.0
       Else (feature 30 > 0.3612145)
        Predict: 6.0
      Else (feature 6 > 0.7879290000000001)
       Predict: 9.0
     Else (feature 27 > 0.3949355)
      If (feature 6 <= 0.7945795)
       Predict: 6.0
      Else (feature 6 > 0.7945795)
       Predict: 9.0

running time：32.0ms

==============================================
test error: 0.46218010554389155
test accuracy: 0.5378198944561084
train accuracy: 0.5375434133933376
desision tree model:
```

```
DecisionTreeClassificationModel (uid=dtc_87e3cfb7372e) of depth 5 with 51 nodes
  If (feature 1 <= 2.1738945000000003)
   If (feature 1 <= -2.9737135)
    If (feature 2 <= 1.4098325)
     If (feature 18 <= 0.7567725000000001)
      Predict: 9.0
     Else (feature 18 > 0.7567725000000001)
      If (feature 2 <= -0.17897000000000002)
       Predict: 9.0
      Else (feature 2 > -0.17897000000000002)
       Predict: 10.0
    Else (feature 2 > 1.4098325)
     If (feature 3 <= -1.8251435)
      If (feature 19 <= -0.12217449999999999)
       Predict: 6.0
      Else (feature 19 > -0.12217449999999999)
       Predict: 10.0
     Else (feature 3 > -1.8251435)
      If (feature 18 <= 0.4248865)
       Predict: 9.0
      Else (feature 18 > 0.4248865)
       Predict: 10.0
   Else (feature 1 > -2.9737135)
    If (feature 1 <= -0.389029)
     If (feature 19 <= -0.3500835)
      If (feature 18 <= 0.371115)
       Predict: 6.0
      Else (feature 18 > 0.371115)
       Predict: 10.0
     Else (feature 19 > -0.3500835)
      If (feature 19 <= 0.6628535)
       Predict: 2.0
      Else (feature 19 > 0.6628535)
       Predict: 7.0
    Else (feature 1 > -0.389029)
     If (feature 19 <= -0.06562599999999999)
      If (feature 1 <= 0.32309)
       Predict: 1.0
      Else (feature 1 > 0.32309)
       Predict: 3.0
     Else (feature 19 > -0.06562599999999999)
      If (feature 19 <= 0.5848705000000001)
       Predict: 8.0
      Else (feature 19 > 0.5848705000000001)
       Predict: 7.0
  Else (feature 1 > 2.1738945000000003)
   If (feature 1 <= 5.7369745000000005)
    If (feature 18 <= -0.44418250000000004)
     If (feature 11 <= -0.6764749999999999)
      If (feature 0 <= -1.21727)
       Predict: 3.0
      Else (feature 0 > -1.21727)
       Predict: 5.0
     Else (feature 11 > -0.6764749999999999)
      If (feature 19 <= 0.157006)
       Predict: 4.0
      Else (feature 19 > 0.157006)
       Predict: 5.0
    Else (feature 18 > -0.44418250000000004)
     If (feature 11 <= 1.237725)
      If (feature 0 <= -5.205881)
       Predict: 4.0
      Else (feature 0 > -5.205881)
       Predict: 5.0
     Else (feature 11 > 1.237725)
      If (feature 19 <= -0.4784385)
       Predict: 4.0
      Else (feature 19 > -0.4784385)
       Predict: 5.0
   Else (feature 1 > 5.7369745000000005)
    If (feature 0 <= 6.8679745)
     Predict: 5.0
    Else (feature 0 > 6.8679745)
     If (feature 1 <= 6.7155795000000005)
      Predict: 3.0
     Else (feature 1 > 6.7155795000000005)
      Predict: 9.0
running time：33.0ms
```