DS/CMPSC 410 Sparing 2021

Instructor: Professor John Yen

TA: Rupesh Prajapati and Dongkuan Xu

Lab 8 Decision Tree Learning Using MLlib and Visualization

The goals of this lab are for you to be able to

- Understand the function of the different steps/stages involved in Spark ML pipeline
- Be able to construct a decision tree using Spark ML machine learning module
- Be able to generate a visualization of Decision Trees
- Be able to compare and evaluate Decision Tree models created using different hyper-parameters (e.g., maximum tree depth)

The data set used in this lab is a Breast Cancer diagnosis dataset.

Submit the following items for Lab 8 (DT)

- Completed Jupyter Notebook of Lab 8 (in HTML or PDF format)
- A word or PDF file that includes the two visualization of decision trees and answers to Exercise 4.

Total Number of Exercises: 4

- Exercise 1: 5 points
- Exercise 2: 5 points
- Exercise 3: 20 points
- Exercise 4: 30 points (Word or PDF file) ## Total Points: 60 points

Due: midnight, March 26 (Friday), 2021

Load and set up the Python files for decision tree visualizations

1. Create a "Lab8DT" directory in the work directory of your ICDS-ROAR home directory.
2. If you have not done so, copy or upload this file to the "Lab8DT" directory.
3. Create a subdirectory under "Lab8DT" called "decision_tree_plot" (named the directory EXACTLY this way).
4. Upload the following three files in Module 8 from Canvas to the decision_tree_plot directory

- decision_tree_parser.py
- decision_tree_plot.py
- tree_template.jinjia2

Follow the instructions below and execute the PySpark code cell by cell below. Make modifications as required.

In [1]:
```
import pyspark
import csv
```

Notice that we use PySpark SQL module to import SparkSession because ML works with SparkSession

Notice also the different methods imported from ML and three submodules of ML: classification, feature, and evaluation.

In [2]:
```
from pyspark import SparkContext
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from pyspark.sql.types import *
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler, I
ndexToString
from pyspark.ml.evaluation import MulticlassClassificationEvaluator, BinaryClas
sificationEvaluator
```

The following two lines import relevant functions from the two python files you uploaded into the decision_tree_plot subdirectory.

In [3]:
```
from decision_tree_plot.decision_tree_parser import decision_tree_parse
from decision_tree_plot.decision_tree_plot import plot_trees
```

This lab runs Spark in the local mode.

Notice we are creating a SparkSession, not a SparkContext, when we use ML pipeline.

The "getOrCreate()" method means we can re-evaluate this without a need to "stop the current SparkSession" first (unlike SparkContext).

In [4]:
```
ss=SparkSession.builder.master("local").appName("lab 8 DT").getOrCreate()
```

As we have seen in Lab 4, SparkSession offers a way to read a CSV/text file with the capability to interpret the first row as being the header and infer the type of different columns based on their values.

Exercise 1: (5 points) Complete the following path with the path for your home directory.

In [5]:
```
data = ss.read.csv("/storage/home/kky5082/ds410/Lab8DT/breast-cancer-wisconsin.
data.txt", header=True, inferSchema=True)
```

Exercise 2: (5 points) Enter your name below:

- My Name: Kangdong Yuan

randomSplit is a method for DataFrame that split data in the DataFrame into two subsets, one for training, the other for testing, using a number as the seed for random number generator.

If you want to generate a different split, you can use a different seed

In [6]:
```
trainingData, testData= data.randomSplit([0.7, 0.3], seed=1234)
```

In [7]:
```
data.printSchema()
```

```
root
 |-- id: integer (nullable = true)
 |-- clump_thickness: integer (nullable = true)
 |-- unif_cell_size: integer (nullable = true)
 |-- unif_cell_shape: integer (nullable = true)
 |-- marg_adhesion: integer (nullable = true)
 |-- single_epith_cell_size: integer (nullable = true)
 |-- bare_nuclei: string (nullable = true)
 |-- bland_chrom: integer (nullable = true)
 |-- norm_nucleoli: integer (nullable = true)
 |-- mitoses: integer (nullable = true)
 |-- class: integer (nullable = true)
```

In [8]:
```
labelIndexer = StringIndexer(inputCol="class", outputCol="indexedLabel").fit(da
ta)
```

In [9]:
```
bnIndexer = StringIndexer(inputCol="bare_nuclei", outputCol="bare_nuclei_index"
).fit(data)
```

In [10]:
```
input_features = ['clump_thickness', 'unif_cell_size', 'unif_cell_shape', 'marg
_adhesion', 'single_epith_cell_size', 'bland_chrom', 'norm_nucleoli', 'mitoses'
, 'bare_nuclei_index']
```

Exercise 3: (10 points) Choose two different values for maxDepth hyperparameter of Decision Trees. (Recommended value for maxDepth: 2 to 7). Run the entire sequence of code below to generate two decision trees.

- Record the f1 measure for each max_depth below
- Make sure you CHANGE the name of the files for saving decision trees (e.g., "DTtree_d3.html", "DTtree_d5.html", ...).

Answer for Exercise 3:

- f1 measure for max_detph 3 = 0.9481309954934459
- f1 measure for max_depth 6 = 0.9519230769230769

In [11]:
```
assembler = VectorAssembler( inputCols=input_features, outputCol="features")
d3=DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="features", maxD
epth=3, minInstancesPerNode=2)
d6=DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="features", maxD
epth=6, minInstancesPerNode=2)
labelConverter = IndexToString(inputCol="indexedLabel", outputCol="predictedCla
ss", labels=labelIndexer.labels)
pipeline_3 = Pipeline(stages=[labelIndexer, bnIndexer, assembler, d3, labelConv
erter])
pipeline_6 = Pipeline(stages=[labelIndexer, bnIndexer, assembler, d6, labelConv
erter])
model_3 = pipeline_3.fit(trainingData)
model_6 = pipeline_6.fit(trainingData)
predictions_3 = model_3.transform(testData)
predictions_6 = model_6.transform(testData)
```

In [12]:
```
predictions_3.persist()
predictions_6.persist()
```

Out[12]:
```
DataFrame[id: int, clump_thickness: int, unif_cell_size: int, unif_cell_shape:
int, marg_adhesion: int, single_epith_cell_size: int, bare_nuclei: string, blan
d_chrom: int, norm_nucleoli: int, mitoses: int, class: int, indexedLabel: doubl
e, bare_nuclei_index: double, features: vector, rawPrediction: vector, probabil
ity: vector, prediction: double, predictedClass: string]
```

In [18]:
```
predictions_3.take(2)
predictions_6.take(2)
```

Out[18]:
```
[Row(id=61634, clump_thickness=5, unif_cell_size=4, unif_cell_shape=3, marg_adh
esion=1, single_epith_cell_size=2, bare_nuclei='?', bland_chrom=2, norm_nucleol
i=3, mitoses=1, class=2, indexedLabel=0.0, bare_nuclei_index=7.0, features=Dens
eVector([5.0, 4.0, 3.0, 1.0, 2.0, 2.0, 3.0, 1.0, 7.0]), rawPrediction=DenseVect
or([0.0, 2.0]), probability=DenseVector([0.0, 1.0]), prediction=1.0, predictedC
lass='2'),
 Row(id=63375, clump_thickness=9, unif_cell_size=1, unif_cell_shape=2, marg_adh
esion=6, single_epith_cell_size=4, bare_nuclei='10', bland_chrom=7, norm_nucleo
li=7, mitoses=2, class=4, indexedLabel=1.0, bare_nuclei_index=1.0, features=Den
seVector([9.0, 1.0, 2.0, 6.0, 4.0, 7.0, 7.0, 2.0, 1.0]), rawPrediction=DenseVec
tor([0.0, 4.0]), probability=DenseVector([0.0, 1.0]), prediction=1.0, predicted
Class='4')]
```

In [13]:
```
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="f1")
```

In [14]:
```
f1 = evaluator.evaluate(predictions_3)
print("f1 depth 3 score:", f1)
f1 = evaluator.evaluate(predictions_6)
print("f1 depth 6 score:", f1)
```

```
f1 depth 3 score: 0.9481309954934459
f1 depth 6 score: 0.9519230769230769
```

stages[3] of the pipeline is "dt" (DecisionTreeClassifier). stages[0] is labelIndexer.

model is a DataFrame representing a trained pipeline.

model.stages[3] gives us the Decision Tree model learned.

In [25]:
```
D3model = model_3.stages[3]
D6model = model_6.stages[3]
print(D3model)
print(D6model)
```

```
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_677ebb58230f, depth
=3, numNodes=11, numClasses=2, numFeatures=9
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_de917f7f69f8, depth
=6, numNodes=29, numClasses=2, numFeatures=9
```

In [27]:
```
model_path="/storage/home/kky5082/ds410/Lab8DT/data"
```

Provide a file name for your decision tree below. If your PySpark code runs successfully, you should see your DT visualization file (e.g., DTtree_d3.html) in your Lab8DT directory.

Note: Change the name of your file before you run the code above with a different maxDepth hyperparameter value.

Download the two decision tree visualization files. Open them with a browser, you will see the trees.

Use screen capture tool (e.g., sreenshot in Mac, snipping tool in PC) to capture the decision trees, and include them in your word document for answers to Exercise 4 (see below)

In [30]:
```
tree=decision_tree_parse(D3model, ss, model_path)
column = dict([(str(idx), i) for idx, i in enumerate(input_features)])
plot_trees(tree, column = column, output_path = 'tree_3.html')
```

In [31]:
```
tree=decision_tree_parse(D6model, ss, model_path)
column = dict([(str(idx), i) for idx, i in enumerate(input_features)])
plot_trees(tree, column = column, output_path = 'tree_6.html')
```

Exercise 4: (30 points) Use a word document to

- (a) Show the decision tree visualization for the two values of max_depth (10 points)
- (b) Discuss the key difference between the two trees. (10 points)
- (c) Discuss which tree you believe is a better model for breast cancer diagnosis, and explain the rationale of your choice. (10 points) ### Submit the PDF version of the word document as a part of this lab.