Solutions_1

April 10, 2019

1 Programming Assignment - 4 A Solution

Use the MLLib API of Spark to construct a decision tree for the Breast Cancer Diagnostic data (we call it dataset1), available from the UC-Irvine ML repository. Select appropriate parameters to generate only a 3-level deep decision tree. Submit the following:

a. Your program code.

b. The choice of parameters and attribute selection metric (Gini index, info gain, etc.) used.

   Impurity Measur: Gini Index

   Max depth of the tree: 3

   Max Bins of the continuous data: [10, 20, 32, 40, 50]

c. Any assumptions made.

   The one which has more Area under the ROC (AUROC) curve is the best model for our application.

   While performing K-fold Cross-validation the value of k is taken as 3.

d. Validation and Train/Test Strategy used.

   Firstly, the entire data is split into train and test. Then, K-fold Cross-validation is performed on train to get model that will not overfit and more generalized. Finally, the model is tested against test data, and the performance metrics are also reported.

e. Decision tree Obtained.

f. Performance shown by the confusion matrix.

In [1]:
```
# Creating a SparkSession

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('DT_BC').getOrCreate()
```

In [2]:
```
# Reading the csv data to a dataframe

df = spark.read.csv('wdbc.csv')
```

```
# Top 1 column of the dataframe

df.head(1)
```

Out[2]: [Row(_c0='842302', _c1='M', _c2='17.99', _c3='10.38', _c4='122.8', _c5='1001', _c6='0.

In [3]:
```
# print schema of the datafame

df.printSchema()
```

```
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: string (nullable = true)
 |-- _c3: string (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: string (nullable = true)
 |-- _c6: string (nullable = true)
 |-- _c7: string (nullable = true)
 |-- _c8: string (nullable = true)
 |-- _c9: string (nullable = true)
 |-- _c10: string (nullable = true)
 |-- _c11: string (nullable = true)
 |-- _c12: string (nullable = true)
 |-- _c13: string (nullable = true)
 |-- _c14: string (nullable = true)
 |-- _c15: string (nullable = true)
 |-- _c16: string (nullable = true)
 |-- _c17: string (nullable = true)
 |-- _c18: string (nullable = true)
 |-- _c19: string (nullable = true)
 |-- _c20: string (nullable = true)
 |-- _c21: string (nullable = true)
 |-- _c22: string (nullable = true)
 |-- _c23: string (nullable = true)
 |-- _c24: string (nullable = true)
 |-- _c25: string (nullable = true)
 |-- _c26: string (nullable = true)
 |-- _c27: string (nullable = true)
 |-- _c28: string (nullable = true)
 |-- _c29: string (nullable = true)
 |-- _c30: string (nullable = true)
 |-- _c31: string (nullable = true)
```

In [4]:
```
# The number of rows

df.count()
```

Out[4]: 569

In [5]:
```
# We need to change the column type of the features to Double as all are read as strin
# Then we drop all the string type columns

from pyspark.sql.types import DoubleType

for i in range(2,len(df.columns)):
    df = df.withColumn("_cc"+str(i),df["_c"+str(i)].cast(DoubleType()))
    df = df.drop('_c'+str(i))

df.head(1)
```

Out[5]: [Row(_c0='842302', _c1='M', _cc2=17.99, _cc3=10.38, _cc4=122.8, _cc5=1001.0, _cc6=0.118

In [6]:
```
# Print the schema of the dataframe

df.printSchema()
```

```
root
 |-- _c0: string (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _cc2: double (nullable = true)
 |-- _cc3: double (nullable = true)
 |-- _cc4: double (nullable = true)
 |-- _cc5: double (nullable = true)
 |-- _cc6: double (nullable = true)
 |-- _cc7: double (nullable = true)
 |-- _cc8: double (nullable = true)
 |-- _cc9: double (nullable = true)
 |-- _cc10: double (nullable = true)
 |-- _cc11: double (nullable = true)
 |-- _cc12: double (nullable = true)
 |-- _cc13: double (nullable = true)
 |-- _cc14: double (nullable = true)
 |-- _cc15: double (nullable = true)
 |-- _cc16: double (nullable = true)
 |-- _cc17: double (nullable = true)
 |-- _cc18: double (nullable = true)
 |-- _cc19: double (nullable = true)
 |-- _cc20: double (nullable = true)
 |-- _cc21: double (nullable = true)
 |-- _cc22: double (nullable = true)
 |-- _cc23: double (nullable = true)
 |-- _cc24: double (nullable = true)
 |-- _cc25: double (nullable = true)
 |-- _cc26: double (nullable = true)
 |-- _cc27: double (nullable = true)
 |-- _cc28: double (nullable = true)
```

```
 |-- _cc29: double (nullable = true)
 |-- _cc30: double (nullable = true)
 |-- _cc31: double (nullable = true)
```

In [7]:
```python
# Drop the first column as well since it is the id of each row which is not a feature

df = df.drop('_c0')

df.head(1)
```

Out[7]:
```
[Row(_c1='M', _cc2=17.99, _cc3=10.38, _cc4=122.8, _cc5=1001.0, _cc6=0.1184, _cc7=0.2776
```

In [9]:
```python
# We will then use StringIndexer to encode the label as Malignant(M):1 and Benign(B):0
# We will also use VectorAssembler to create a vector of features which will be given
# The above two stages are combined to form a Pipeline

from pyspark.ml.feature import StringIndexer, VectorAssembler

label_stringIdx = StringIndexer(inputCol = '_c1', outputCol = 'label')

assemblerInputs = ["_cc"+str(i) for i in range(2,len(df.columns))]

assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")

stages=[label_stringIdx, assembler]
```

In [10]:
```python
# Create a Pipeline and Pass the dataframe into the pipeline to transform as required

cols = df.columns

from pyspark.ml import Pipeline

pipeline = Pipeline(stages=stages)

pipelineModel = pipeline.fit(df)

df = pipelineModel.transform(df)

n_cols = ['label', 'features']+cols

df = df.select(n_cols)

df.printSchema()
```

```
root
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
```

```
 |-- _c1: string (nullable = true)
 |-- _cc2: double (nullable = true)
 |-- _cc3: double (nullable = true)
 |-- _cc4: double (nullable = true)
 |-- _cc5: double (nullable = true)
 |-- _cc6: double (nullable = true)
 |-- _cc7: double (nullable = true)
 |-- _cc8: double (nullable = true)
 |-- _cc9: double (nullable = true)
 |-- _cc10: double (nullable = true)
 |-- _cc11: double (nullable = true)
 |-- _cc12: double (nullable = true)
 |-- _cc13: double (nullable = true)
 |-- _cc14: double (nullable = true)
 |-- _cc15: double (nullable = true)
 |-- _cc16: double (nullable = true)
 |-- _cc17: double (nullable = true)
 |-- _cc18: double (nullable = true)
 |-- _cc19: double (nullable = true)
 |-- _cc20: double (nullable = true)
 |-- _cc21: double (nullable = true)
 |-- _cc22: double (nullable = true)
 |-- _cc23: double (nullable = true)
 |-- _cc24: double (nullable = true)
 |-- _cc25: double (nullable = true)
 |-- _cc26: double (nullable = true)
 |-- _cc27: double (nullable = true)
 |-- _cc28: double (nullable = true)
 |-- _cc29: double (nullable = true)
 |-- _cc30: double (nullable = true)
 |-- _cc31: double (nullable = true)
```

In [11]:
```
# Top row of the dataframe

df.head(1)
```

Out[11]:
```
[Row(label=1.0, features=DenseVector([17.99, 10.38, 122.8, 1001.0, 0.1184, 0.2776, 0.3
```

In [12]:
```
# Spliting the data into train and split

train, test = df.randomSplit([0.7, 0.3], seed = 2018)

# Print the top of the train and test dataframe

print(train.head(1))

print(test.head(1))
```

```
[Row(label=0.0, features=DenseVector([6.981, 13.43, 43.79, 143.5, 0.117, 0.0757, 0.0, 0.0, 0.19
[Row(label=0.0, features=DenseVector([7.76, 24.54, 47.92, 181.0, 0.0526, 0.0436, 0.0, 0.0, 0.15
```

In [14]:
```python
# We will use CrossValidation to to tune the maxBins size to get a best model
# The evaluator to this will be the Area under the ROC curve(AUROC)

from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator() # The evalutor is AUROC by default

# Impurity is Gini by default

dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label', maxDepth = 3

# We will add the maxBin parameters based which the CrossValidation will go

paramGrid = ParamGridBuilder() \
    .addGrid(dt.maxBins, [10, 20, 32, 40, 50]) \
    .build()

cv = CrossValidator(estimator=dt, estimatorParamMaps=paramGrid, evaluator=evaluator, n

# Fit the train data to get the model

cvModel = cv.fit(train)
```

In [15]:
```python
# Pull out the best model

BestModel = cvModel.bestModel
print(BestModel.toDebugString)
```

```
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_7311e4b22bcd) of depth 3 with 13 no
  If (feature 20 <= 17.1)
   If (feature 27 <= 0.1572)
    Predict: 0.0
   Else (feature 27 > 0.1572)
    If (feature 21 <= 23.515)
     Predict: 0.0
    Else (feature 21 > 23.515)
     Predict: 1.0
  Else (feature 20 > 17.1)
   If (feature 26 <= 0.19465)
    If (feature 1 <= 19.509999999999998)
     Predict: 0.0
    Else (feature 1 > 19.509999999999998)
```

```
     Predict: 1.0
   Else (feature 26 > 0.19465)
    If (feature 11 <= 0.48614999999999997)
     Predict: 0.0
    Else (feature 11 > 0.48614999999999997)
     Predict: 1.0
```

In [16]:
```python
# What are the Parameters of the best model

print(BestModel.explainParams())
```

```
cacheNodeIds: If false, the algorithm will pass trees to executors to match instances with node
checkpointInterval: set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means th
featuresCol: features column name (default: features, current: features)
impurity: Criterion used for information gain calculation (case-insensitive). Supported options
labelCol: label column name (default: label, current: label)
maxBins: Max number of bins for discretizing continuous features.  Must be >=2 and >= number of
maxDepth: Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 in
maxMemoryInMB: Maximum memory in MB allocated to histogram aggregation. (default: 256)
minInfoGain: Minimum information gain for a split to be considered at a tree node. (default: 0
minInstancesPerNode: Minimum number of instances each child must have after split.  If a split
predictionCol: prediction column name (default: prediction)
probabilityCol: Column name for predicted class conditional probabilities. Note: Not all models
rawPredictionCol: raw prediction (a.k.a. confidence) column name (default: rawPrediction)
seed: random seed (default: 956191873026065186)
thresholds: Thresholds in multi-class classification to adjust the probability of predicting ea
```

In [17]:
```python
# Evaluation of the model on the test data and the following are computed:
# 1. Precision of 1
# 2. Precision of 0
# 3. Recall of 1
# 4. Recall of 0
# 5. F-1 Score
# 6. Confusion Matrix

from pyspark.mllib.evaluation import MulticlassMetrics

def getPredictionsLabels(model, test_data):
    predictions = cvModel.transform(test_data)
    predictions_n = predictions["label", "prediction"]
    predictionsAndLabels = predictions_n.rdd.map(tuple)
    return  predictionsAndLabels

def printMetrics(predictions_and_labels):
    metrics = MulticlassMetrics(predictions_and_labels)
```

```python
    print ('Precision of 1:Malignant\t', metrics.precision(1))
    print ('Precision of 0:Benign\t\t', metrics.precision(0))
    print ('Recall of 1:Malignant\t\t', metrics.recall(1))
    print ('Recall of 0:Benign\t\t', metrics.recall(0))
    print ('F-1 Score\t\t\t', metrics.fMeasure())
    print ('Confusion Matrix\n', metrics.confusionMatrix().toArray())

predictions_and_labels = getPredictionsLabels(cvModel, test)

printMetrics(predictions_and_labels)
```

```
Precision of 1:Malignant         0.863013698630137
Precision of 0:Benign                 0.9900990099009901
Recall of 1:Malignant                 0.984375
Recall of 0:Benign                  0.9090909090909091
F-1 Score                          0.9367816091954023
Confusion Matrix
 [[100.  10.]
 [  1.  63.]]
```