**M10 Lab: Building the ML Pipeline in Spark ML**

Q1. Summarize problem-specific parameters and what each parameter means.

These Problem specific parameters are the problem we want to solve and our dataset. They should be specific, and they do not need any tunning.

**Algorithm**: This specifies the type of decision tree, either for classification or for regression.

**NumClassess:** This Specify the number of Classes. (For Classification Problems only).

**CategoricalFeaturesInfo:** Specifies which features are categorical and how many categorical values each of the features can take. This gives the map of feature indices to feature arity. Any features not in this map is considered as continuous.

For example, the Map of (0-> 2, 4->10) specify the feature 0 is binary taking values 0 and 1

and feature 4 has feature values varies from 0,1,2,3…9.

feature indices are 0-based: features 0 and 4 are the 1st and 5th elements of an instance’s feature vector.

We do not need to specify CategoricalFeatureInfo. The algorithm still run and get reasonable results.

Performance will be better if Categorical features are properly specified.

Q2. Summarize tunable parameters and what each parameter means.

These parameters may be tuned, but we need to be careful about overfitting.

**maxBins :** Number of bins used for approximating the continuous features.

Increasing maxbins allows the algorithm to consider the more split candidates and make fine-grained split decision, However, it also increases computation and communication.

The max bin must be at least the number of categories M in Categorical features.

**maxMemoryInMB** : Amount of memory used for collecting sufficient statistics.

The default values is 256 MB to allow the decision algorithm to work in most of the scenarios.

Increasing the maxMemoryInMB allows faster training by allowing the fewer passes over the data.  there may be decreasing returns as maxMemoryInMB grows since the amount of communication on each iteration can be proportional to maxMemoryInMB.  For faster processing, the decision tree algorithm collects statistics about groups of nodes to split (rather than 1 node at a time). The number of nodes which can be handled in one group is determined by the memory requirements (which vary per features).

**subsamplingRate** : Fraction of training data used for training the decision tree . The Parameter is useful when training the group of the trees, where it can be used for subsampling the original data.

**Impurity**: Measure used to choose the split candidates, this parameter must matches the algo parameter.

Q3. Using your databricks account, run the classification code on the dataset file provided. List the obtained results. ***(Make sure to comment out the last two lines of code that save & load the model.)***

from pyspark.mllib.tree import DecisionTree, DecisionTreeModel

from pyspark.mllib.util import MLUtils

# Load and parse the data file into an RDD of LabeledPoint.

data = MLUtils.loadLibSVMFile(sc, 'dbfs:/FileStore/tables/colon\_cancer\_01.txt')

# Split the data into training and test sets (30% held out for testing)

(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a DecisionTree model.

# Empty categoricalFeaturesInfo indicates all features are continuous.

model = DecisionTree.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},

impurity='gini', maxDepth=5, maxBins=32)

# Evaluate model on test instances and compute test error

predictions = model.predict(testData.map(lambda x: x.features))

labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)

testErr = labelsAndPredictions.filter(

lambda lp: lp[0] != lp[1]).count() / float(testData.count())

print('Test Error = ' + str(testErr))

print('Learned classification tree model:')

print(model.toDebugString())

A screenshot of a computer

Description automatically generated

from pyspark.mllib.tree import DecisionTree, DecisionTreeModel

from pyspark.mllib.util import MLUtils

# Load and parse the data file into an RDD of LabeledPoint.

data = MLUtils.loadLibSVMFile(sc, 'dbfs:/FileStore/tables/colon\_cancer\_01.txt')

# Split the data into training and test sets (30% held out for testing)

(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a DecisionTree model.

# Empty categoricalFeaturesInfo indicates all features are continuous.

model = DecisionTree.trainRegressor(trainingData, categoricalFeaturesInfo={},

impurity='variance', maxDepth=5, maxBins=32)

# Evaluate model on test instances and compute test error

predictions = model.predict(testData.map(lambda x: x.features))

labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)

testMSE = labelsAndPredictions.map(lambda lp: (lp[0] - lp[1]) \* (lp[0] - lp[1])).sum() /\

float(testData.count())

print('Test Mean Squared Error = ' + str(testMSE))

print('Learned regression tree model:')

print(model.toDebugString())

Graphical user interface, text, application

Description automatically generated

Q4. Change any of the tunable attributes in the code. For example,

* Change the ***impurity***function from ***'gini'*** to ***'entropy'***.
* Change the ***maxDept***to ***10***then ***20***.

from pyspark.mllib.tree import DecisionTree, DecisionTreeModel

from pyspark.mllib.util import MLUtils

# Load and parse the data file into an RDD of LabeledPoint.

data = MLUtils.loadLibSVMFile(sc, 'dbfs:/FileStore/tables/colon\_cancer\_01.txt')

# Split the data into training and test sets (30% held out for testing)

(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a DecisionTree model.

# Empty categoricalFeaturesInfo indicates all features are continuous.

model = DecisionTree.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},

impurity='entropy', maxDepth=10, maxBins=32)

# Evaluate model on test instances and compute test error

predictions = model.predict(testData.map(lambda x: x.features))

labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)

testErr = labelsAndPredictions.filter(

lambda lp: lp[0] != lp[1]).count() / float(testData.count())

print('Test Error = ' + str(testErr))

print('Learned classification tree model:')

print(model.toDebugString())

# Save and load model

#model.save(sc, "dbfs:/FileStore/table/")

#sameModel = DecisionTreeModel.load(sc, "dbfs:/FileStore/tables/")

Graphical user interface, text, application

Description automatically generated

from pyspark.mllib.tree import DecisionTree, DecisionTreeModel

from pyspark.mllib.util import MLUtils

# Load and parse the data file into an RDD of LabeledPoint.

data = MLUtils.loadLibSVMFile(sc, 'dbfs:/FileStore/tables/colon\_cancer\_01.txt')

# Split the data into training and test sets (30% held out for testing)

(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a DecisionTree model.

# Empty categoricalFeaturesInfo indicates all features are continuous.

model = DecisionTree.trainClassifier(trainingData, numClasses=2, categoricalFeaturesInfo={},

impurity='entropy', maxDepth=20, maxBins=32)

# Evaluate model on test instances and compute test error

predictions = model.predict(testData.map(lambda x: x.features))

labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)

testErr = labelsAndPredictions.filter(

lambda lp: lp[0] != lp[1]).count() / float(testData.count())

print('Test Error = ' + str(testErr))

print('Learned classification tree model:')

print(model.toDebugString())

# Save and load model

#model.save(sc, "dbfs:/FileStore/tab/")

#sameModel = DecisionTreeModel.load(sc, "dbfs:/FileStore/tables/")

Graphical user interface, text, application

Description automatically generated

* Did the test error result change? Report all obtained test errors along with the used parameter values.

Yes , test Error changed . impurity='gini', maxDepth=10, maxBins=32, numClasses=2, categoricalFeaturesInfo={} these are parameter used. . Test Error = 0.35714285714285715 and

When maxDepth= 20 ,Test Error = 0.25.

### **Part II: Decision Tree Training & Selection with K-Fold Cross Validation**

from pyspark.ml import Pipeline

from pyspark.ml.classification import DecisionTreeClassifier

from pyspark.ml.feature import StringIndexer, VectorIndexer

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

# Load the data stored in LIBSVM format as a DataFrame.

data = spark.read.format("libsvm").load("dbfs:/FileStore/tables/colon\_cancer\_01.txt")

# Index labels, adding metadata to the label column.

# Fit on whole dataset to include all labels in index.

labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)

# Automatically identify categorical features, and index them.

# We specify maxCategories so features with > 4 distinct values are treated as

#continuous.

featureIndexer =VectorIndexer(inputCol="features", outputCol="indexedFeatures",

maxCategories=4).fit(data)

# Split the data into training and test sets (30% held out for testing)

(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a DecisionTree model.

dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")

# Chain indexers and tree in a Pipeline

pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dt])

# Select (prediction, true label) and compute test error

evaluator1 = MulticlassClassificationEvaluator(

labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")

paramGrid = ParamGridBuilder() \

.addGrid(dt.maxBins, [2, 5, 10, 20]).build()

#.addGrid(dt.impurity, ['gini','entropy']) \

crossval = CrossValidator(estimator=pipeline,

estimatorParamMaps=paramGrid,

evaluator=evaluator1,

numFolds=3) # use 3+ folds in practice

# Train model. This also runs the indexers.

cvModel = crossval.fit(trainingData)

# Make predictions.

predictions = cvModel.transform(testData)

# Select example rows to display.

predictions.select("prediction", "indexedLabel", "features").show(5)

accuracy = evaluator1.evaluate(predictions)

print("Test Error = %g " % (1.0 - accuracy))

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

1. What is the obtained test error?

0.375

from pyspark.ml import Pipeline

from pyspark.ml.classification import DecisionTreeClassifier

from pyspark.ml.feature import StringIndexer, VectorIndexer

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

# Load the data stored in LIBSVM format as a DataFrame.

data = spark.read.format("libsvm").load("dbfs:/FileStore/tables/colon\_cancer\_01.txt")

# Index labels, adding metadata to the label column.

# Fit on whole dataset to include all labels in index.

labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)

# Automatically identify categorical features, and index them.

# We specify maxCategories so features with > 4 distinct values are treated as

#continuous.

featureIndexer =VectorIndexer(inputCol="features", outputCol="indexedFeatures",

maxCategories=4).fit(data)

# Split the data into training and test sets (30% held out for testing)

(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a DecisionTree model.

dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")

# Chain indexers and tree in a Pipeline

pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dt])

# Select (prediction, true label) and compute test error

evaluator1 = MulticlassClassificationEvaluator(

labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")

paramGrid = ParamGridBuilder() \

.addGrid(dt.maxBins, [2, 5, 10, 20]).addGrid(dt.impurity, ['gini','entropy']).build()

crossval = CrossValidator(estimator=pipeline,

estimatorParamMaps=paramGrid,

evaluator=evaluator1,

numFolds=3) # use 3+ folds in practice

# Train model. This also runs the indexers.

cvModel = crossval.fit(trainingData)

# Make predictions.

predictions = cvModel.transform(testData)

# Select example rows to display.

predictions.select("prediction", "indexedLabel", "features").show(5)

accuracy = evaluator1.evaluate(predictions)

print("Test Error = %g " % (1.0 - accuracy))

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

What are the obtained results?

0.2