

Lab - 15

Building the ML Pipeline in Spark ML

HARSH SIDDHAPURA

1230169813

04/14/2024

“PART I”

“The different parameters of a decision tree classifier”

1. “Summarize problem-specific parameters and what each parameter means” (Canvas)

They could be summarized as follows:

- **algo:** This option controls the precise type of decision tree algorithm used. The choices are 'Classification' or 'Regression'. 'Classification' anticipates a discrete label, such as whether an email is spam. On the other hand, 'regression' is employed when the purpose is to predict a continuous value, such as estimating the price of a property based on multiple factors.
- **numClasses:** This parameter only pertains to classification issues. It represents the total number of different class labels found in your collection. For example, in a binary classification task like spam detection, numClasses would be set to two, indicating the two potential outcomes: spam or non-spam.
- **categoricalFeaturesInfo:** This parameter is a map that specifies which characteristics in your dataset are categorical and how many categories each of these features can have. The map's key is the feature index, and its value is the feature arity, or number of categories. For example, Map(0 -> 2, 4 -> 10) specifies that the first feature (index 0) is binary (with potential values of 0 or 1), whereas the fifth feature (index 4) includes ten categories (with values ranging from 0 to 9). Features not included in this map are considered continuous. It's worth mentioning that, while specifying categoricalFeaturesInfo is optional, it can improve algorithm speed by ensuring categorical features are handled correctly.

2. “Summarize tunable parameters and what each parameter means” (Canvas)

The tunable parameters play a significant role in optimizing the performance of decision tree algorithms. They are summarized as follows:

- **maxBins:** This parameter represents the number of bins to discretize continuous features. A greater maxBins number allows the algorithm to consider more possible splits, resulting in more nuanced split judgments. However, this increases the computational and communication needs. It is critical to remember that maxBins

should be at least as large as the maximum number of categories in each category feature.

- **maxMemoryInMB:** This option specifies the RAM allotment for collecting suitable statistics. The default option is 256 MB, which ensures that the decision tree method works successfully in most cases. Increasing maxMemoryInMB can speed up training by lowering the amount of data passes if enough memory is available. However, the benefits may decrease as maxMemoryInMB grows due to the proportional connection between maxMemoryInMB and the communication required for every iteration.
- **subsamplingRate:** This value represents the proportion of training data used for decision tree learning. It's especially important when training tree ensembles (like RandomForest and GradientBoostedTrees), where subsampling the original data might help. This value is less important when training a single decision tree since the number of training examples is usually not the key limitation.
- **impurity:** This parameter refers to the impurity measure used to pick amongst candidate splits. It must be consistent with the algo parameter. For classification tasks, the impurity measure might be Gini or Entropy, or another measure, depending on the method. In regression tasks, variance reduction is a typical impurity metric.

3. “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.)” (Canvas)

Code:

```
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,
    '/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)
testError = labelsAndPredictions.filter(
    lambda lp: lp[0] != lp[1]).count() / float(testData.count())
print('Test Error = ' + str(testError))
print('Learned classification tree model:')
print(model.toDebugString())
```

Result:

```
Test Error = 0.2631578947368421
Learned classification tree model:
DecisionTreeModel classifier of depth 2 with 7 nodes
  If (feature 244 <= 0.32059)
    If (feature 44 <= 1.05736)
      Predict: 0.0
    Else (feature 44 > 1.05736)
      Predict: 1.0
  Else (feature 244 > 0.32059)
    If (feature 140 <= 0.3559515)
      Predict: 1.0
    Else (feature 140 > 0.3559515)
      Predict: 0.0
```

The result can be explained below :

- **Test Error:** The test error is calculated as the fraction of test instances that were classified incorrectly by the model. In this case, the test error is approximately 0.263, meaning the model misclassified about 26.3% of the test instances.
- **Learned Classification Tree Model:** The model learned by the decision tree algorithm is printed in a human-readable format. This simple decision tree model has 2 and 7 nodes in depth. Here's how to interpret the model:

- If the value of feature 244 is less than or equal to 0.32059, the model checks the value of feature 44. If feature 44 is less than or equal to 1.05736, the model predicts the class label 0.0. Otherwise, if feature 44 is greater than 1.05736, the model predicts the class label 1.0.
- If the value of feature 244 is greater than 0.32059, the model checks the value of feature 140. If feature 140 is less than or equal to 0.3559515, the model predicts the class label 1.0. Otherwise, if feature 140 is greater than 0.3559515, the model predicts the class label 0.0.

The features (244, 44, and 140) and their thresholds (0.32059, 1.05736, and 0.3559515) were learned from the training data using the Gini impurity measure. The class labels (0.0 and 1.0) represent the two classes of the target variable in your colon cancer dataset. The exact meaning of these features and class labels would depend on the specifics of your dataset.

Name: Harsh Siddhapura
ASU ID: 1230169813

```

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, '/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())

```

Test Error = 0.2631578947368421
 Learned classification tree model:
 DecisionTreeModel classifier of depth 2 with 7 nodes
 If (feature 244 <= 0.32059)
 If (feature 44 <= 1.05736)
 Predict: 0.0
 Else (feature 44 > 1.05736)
 Predict: 1.0
 Else (feature 244 > 0.32059)
 If (feature 140 <= 0.3559515)
 Predict: 1.0
 Else (feature 140 > 0.3559515)
 Predict: 0.0

4. “Change any of the tunable attributes in the code. For example,”

- Change the impurity function from 'gini' to 'entropy'.” (Canvas)

```
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, '/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=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())
```

The screenshot shows a Databricks workspace interface. The top navigation bar includes 'Data Science & Eng...', 'Workspace', 'Recents', 'Search', 'Catalog', 'Compute', and 'Workflows'. The main area displays a notebook titled 'Name: Harsh Siddhapura' and 'ASU ID: 1230169813'. The notebook is in 'Python' mode and shows the same code as the previous block. The output of the code is visible, showing the test error and the learned classification tree model.

```
Test Error = 0.2692307692307692
Learned classification tree model:
DecisionTreeModel classifier of depth 3 with 7 nodes
If (feature 376 <= -0.17097600000000002)
  Predict: 0.0
Else (feature 376 > -0.17097600000000002)
  If (feature 355 <= 0.7066235000000001)
    Predict: 1.0
  Else (feature 355 > 0.7066235000000001)
    If (feature 4 <= -0.3166135)
      Predict: 1.0
    Else (feature 4 > -0.3166135)
      Predict: 0.0
```

- “Change the maxDepth to 10 then 20” (Canvas)

```
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, '/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())
```

The screenshot displays the Databricks community cloud interface. At the top, a browser window shows the URL: `community.cloud.databricks.com/?o=3431424865199059#notebook/118479515793372/command/118479515793373`. Below the browser, the Databricks workspace is visible. The notebook is titled "Name: Harsh Siddhapura" and "ASU ID: 1230169813". The notebook is named "ML-Pipeline" and is written in Python. The code in the notebook is identical to the code block above. The execution output shows the test error and the learned classification tree model. The output is as follows:

```
Test Error = 0.25
Learned classification tree model:
DecisionTreeModel classifier of depth 2 with 5 nodes
If (feature 1634 <= 0.1587695)
  Predict: 0.0
Else (feature 1634 > 0.1587695)
  If (feature 145 <= -0.9622945)
    Predict: 0.0
  Else (feature 145 > -0.9622945)
    Predict: 1.0
```

```

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, '/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)
testError = labelsAndPredictions.filter(
    lambda lp: lp[0] != lp[1]).count() / float(testData.count())
print('Test Error = ' + str(testError))
print('Learned classification tree model:')
print(model.toDebugString())

```

The screenshot displays a Databricks notebook interface. At the top, the browser address bar shows the URL: `community.cloud.databricks.com/?o=3431424865199059#notebook/118479515793372/command/118479515793373`. The notebook is titled "ML-Pipeline" and is owned by "harshsiddhapura300@gmail.com". The user's name, "Name: Harsh Siddhapura", and ASU ID, "ASU ID: 1230169813", are displayed prominently. The notebook contains a Python script that trains a Decision Tree classifier on colon cancer data. The script uses `pyspark.mllib` to load the data, split it into training and testing sets, train the model, and evaluate its performance. The output of the script is shown in a code cell, indicating a test error of 0.1875 and providing the learned classification tree model structure.

```

Name: Harsh Siddhapura
ASU ID: 1230169813

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, '/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)
testError = labelsAndPredictions.filter(
    lambda lp: lp[0] != lp[1]).count() / float(testData.count())
print('Test Error = ' + str(testError))
print('Learned classification tree model:')
print(model.toDebugString())

```

Test Error = 0.1875
Learned classification tree model:
DecisionTreeModel classifier of depth 3 with 7 nodes
If (feature 376 <= -0.208317)
Predict: 0.0
Else (feature 376 > -0.208317)
If (feature 626 <= 0.1742205)
If (feature 787 <= -1.207635)
Predict: 0.0
Else (feature 787 > -1.207635)
Predict: 1.0
Else (feature 626 > 0.1742205)
Predict: 0.0

- **“Did the test error result change? Report all obtained test errors along with the used parameter values.” (Canvas)**

The test error results are as follows:

- impurity='gini', maxDepth=5:
Test Error: 0.26
- impurity='entropy', maxDepth=5:
Test Error: 0.26
- impurity='entropy', maxDepth=10:
Test Error: 0.25
- impurity='entropy', maxDepth=20:
Test Error: 0.187

The results demonstrate the impact of varying the impurity and maxDepth parameters on the performance of the decision tree model:

- Utilizing the Gini impurity measure with a maximum tree depth of 5, the model resulted in a test error 0.26. This implies that the model incorrectly classified 26% of the instances in the test set.
- Switching to the Entropy impurity measure, while keeping the maximum tree depth at 5, the test error remained at 0.26, suggesting comparable performance between the Gini and Entropy impurity measures at this tree depth.
- Elevating the maxDepth to 10 while using the Entropy impurity measure led to a marginal decrease in the test error to 0.25. This indicates that a deeper decision tree slightly improved the model's performance.
- However, further augmenting the maxDepth to 20 with the Entropy impurity measure resulted in a decreased test error of 0.187.

These findings underscore the significance of parameter tuning in decision tree models. While deeper trees can encapsulate more complex patterns, they also risk overfitting. Similarly, the choice of impurity measure can influence the model's performance, and the optimal choice may depend on the specific characteristics of your dataset and task.

“PART II”

“Decision Tree Training & Selection with K-Fold Cross Validation”

The screenshot shows a Databricks ML-Pipeline notebook titled "ML-Pipeline" with the language set to Python. The notebook is owned by Harsh Siddhapura (ASU ID: 1230169813). The code defines a pipeline for training a Decision Tree Classifier on a dataset stored in a table named "colon_cancer_01.txt". The pipeline includes a StringIndexer for the label column, a VectorIndexer for the features, and a DecisionTreeClassifier. The data is split into training and test sets (30% held out for testing). The model is trained using cross-validation with 3 folds. The final output shows the test error rate as 0.263158.

```
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("/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()

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))
```

The screenshot shows the same Databricks ML-Pipeline notebook, but now displaying the results of the training process. The code has been executed, and the output shows the test error rate as 0.263158. The Spark Jobs output displays the predicted values for the first 5 rows of the test data.

```
# 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))
```

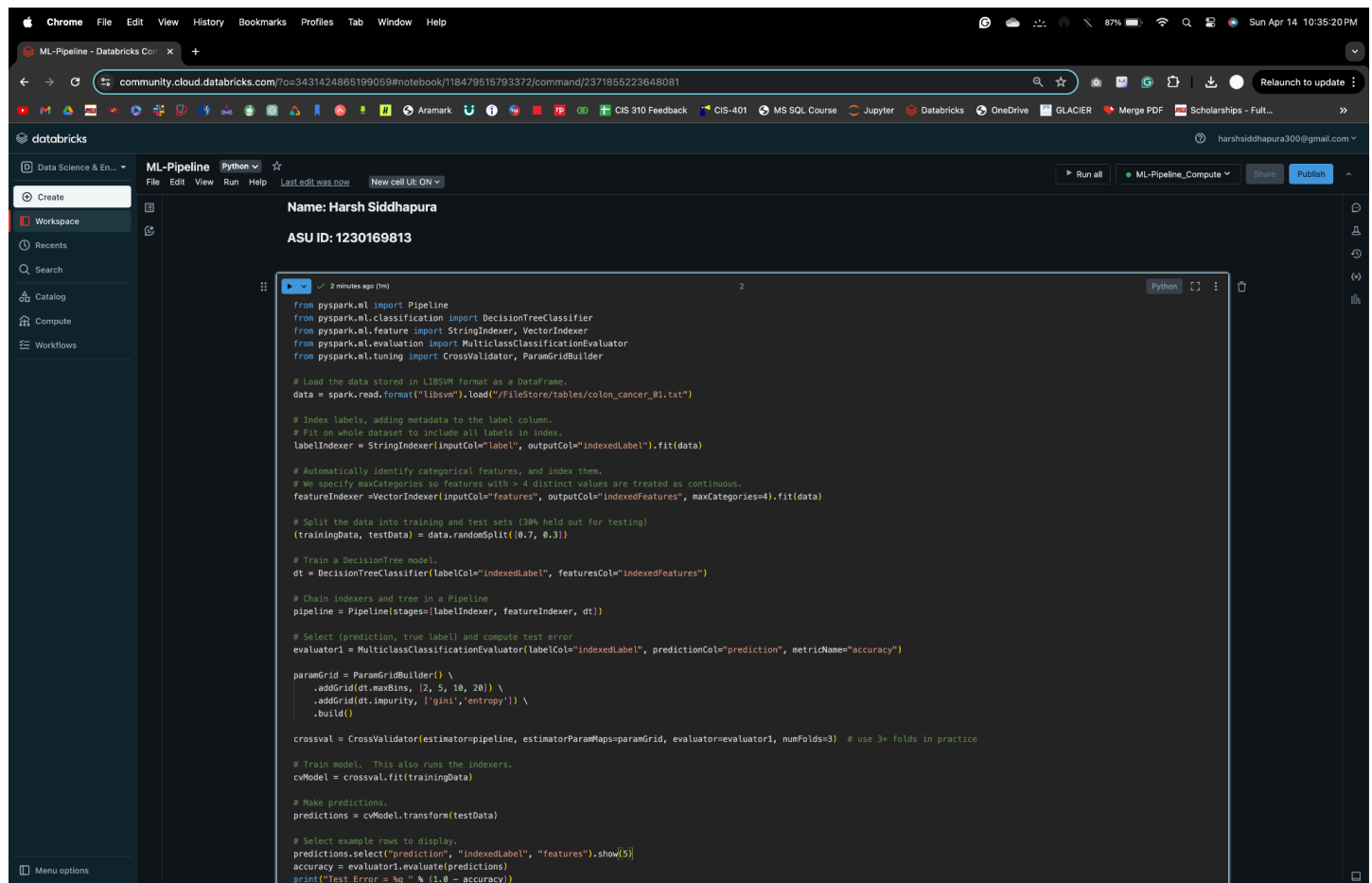
(12) Spark Jobs

prediction	indexedLabel	features
0.0	0.0	(2000, [0, 1, 2, 3, 4, ...])
0.0	0.0	(2000, [0, 1, 2, 3, 4, ...])
0.0	0.0	(2000, [0, 1, 2, 3, 4, ...])
0.0	0.0	(2000, [0, 1, 2, 3, 4, ...])
0.0	0.0	(2000, [0, 1, 2, 3, 4, ...])

only showing top 5 rows

Test Error = 0.263158

Putting back the removed line to insert another grid



```
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("/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 to 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")

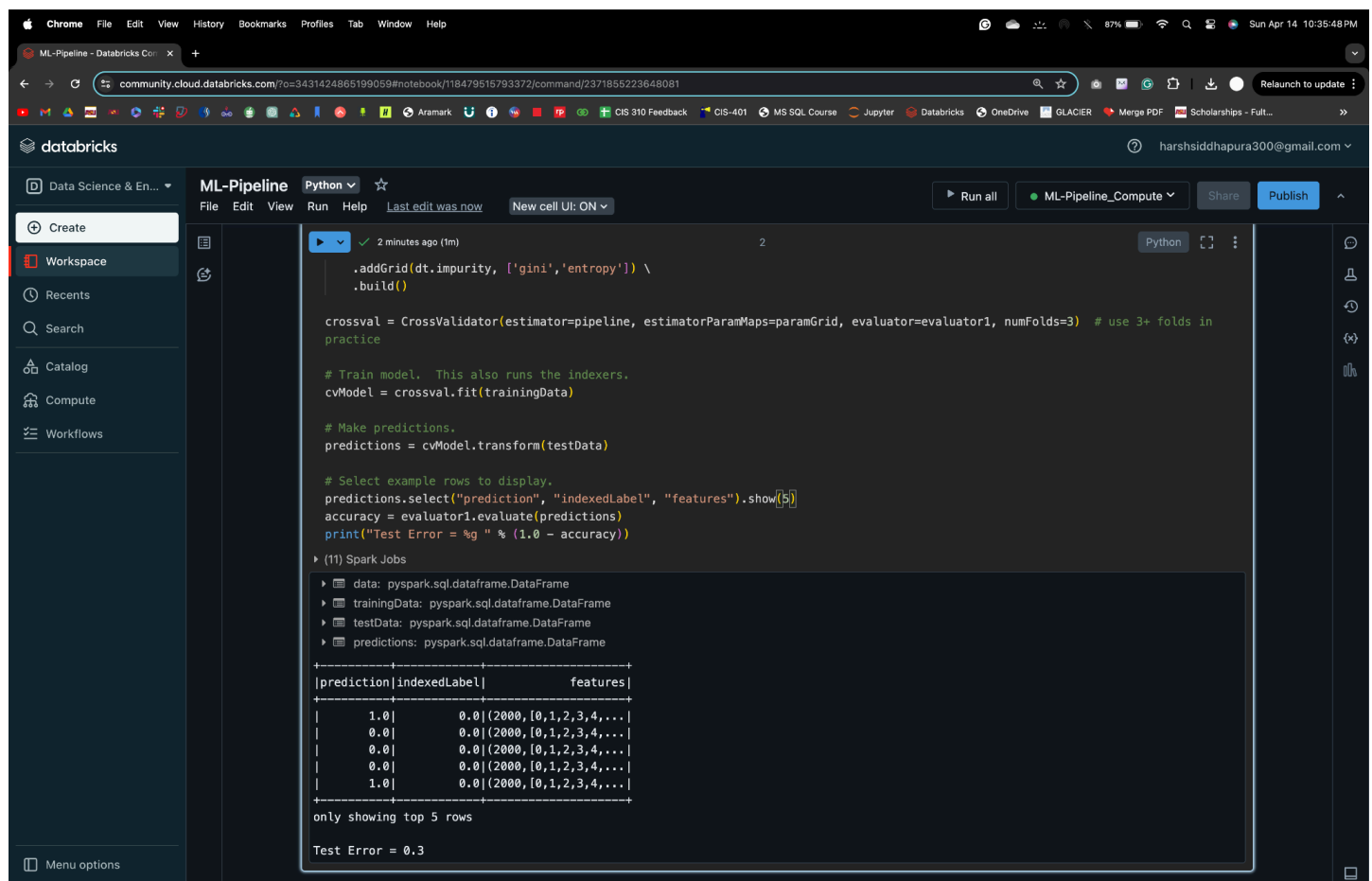
paramGrid = ParamGridBuilder() \
    .addGrid(dt.maxBins, [2, 5, 10, 20]) \
    .addGrid(dt.impurity, ['gini', 'entropy']) \
    .build()

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))
```



```
.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))
```

▶ (11) Spark Jobs

- data: pyspark.sql.dataframe.DataFrame
- trainingData: pyspark.sql.dataframe.DataFrame
- testData: pyspark.sql.dataframe.DataFrame
- predictions: pyspark.sql.dataframe.DataFrame

prediction	indexedLabel	features
1.0	0.0	(2000, [0, 1, 2, 3, 4, ...])
0.0	0.0	(2000, [0, 1, 2, 3, 4, ...])
0.0	0.0	(2000, [0, 1, 2, 3, 4, ...])
0.0	0.0	(2000, [0, 1, 2, 3, 4, ...])
1.0	0.0	(2000, [0, 1, 2, 3, 4, ...])

only showing top 5 rows

Test Error = 0.3