



# CIS5560 Term Project Tutorial



**Authors:** Tanvi Gawade, Swarnim Jambhule, Sushant Burde

**Instructor:** [Jongwook Woo](#)

**Date:** 05/19/2019

## Lab Tutorial

Tanvi ([tgawade@calstatela.edu](mailto:tgawade@calstatela.edu))

Swarnim ([sjambhu@calstatela.edu](mailto:sjambhu@calstatela.edu))

Sushant ([sburde@calstatela.edu](mailto:sburde@calstatela.edu))

05/19/2019

## Classification for Floating or Non-Floating Items from the Library Inventory On Data Bricks in Spark Machine Learning

### Objectives

List what your objectives are. In this hands-on lab, you will learn how to:

- Get data manually
- Create Spark cluster

- Writing PySpark codes to develop a predictive model.
- Classification of Floating and Non-Floating values using Random Forest Classification and Gradient Boosting Tree Classification.
- Visualization

## Platform Spec

- Data Bricks PySpark
- Databricks Runtime Version: 5.2(Incl. Apache Spark 2.4.0, Scala 2.11)
- Execution: Single Node
- Memory: 6GB Capacity

## Step 1: Creating a Cluster in Data Bricks

1. This step is to create a cluster for the execution of the codes.

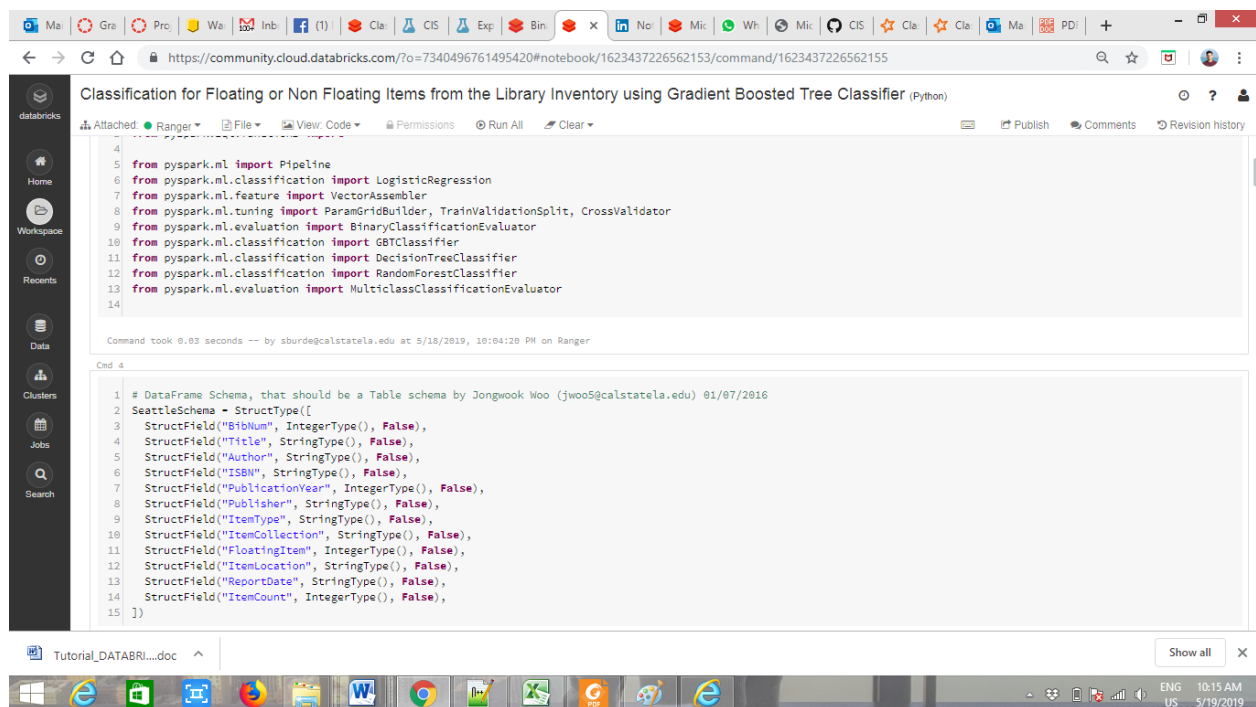
The screenshot shows the Databricks Clusters page in a web browser. The URL is <https://community.cloud.databricks.com/?o=7340496761495420#setting/clusters>. The page has a sidebar with navigation options: Home, Workspace, Recents, Data, Clusters (selected), Jobs, and Search. The main content area is titled 'Clusters' and includes a '+ Create Cluster' button. Below this, there are two sections: 'Interactive Clusters' and 'Job Clusters'. The 'Interactive Clusters' section shows a table with 2 clusters, 0 pinned. The table has columns: Name, State, Nodes, Driver, Worker, Runtime, Creator, and Actions. The first cluster is 'Ranger' with state 'Running', 1 (0 spot) nodes, and runtime '5.3 (includes Apa...'. The second cluster is 'Ranger' with state 'Terminated', 0 nodes, and runtime '5.3 (includes Apa...'. The 'Job Clusters' section shows 'No clusters found'.

Name	State	Nodes	Driver	Worker	Runtime	Creator	Actions
Ranger	Running	1 (0 spot)	Community Opt...	Community Opt...	5.3 (includes Apa...	sburde@calstat...	...
Ranger	Terminated	-	Community Opt...	Community Opt...	5.3 (includes Apa...	sburde@calstat...	...

Properties: -

- Python Version: 2
- Driver Type: Community Optimized
- Availability Zone: us-west-2c

## Step 2: Prepare the Data



The screenshot displays a Databricks notebook titled "Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)". The interface includes a sidebar with navigation options like Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main area shows a code editor with the following Python code:

```
4 from pyspark.ml import Pipeline
5 from pyspark.ml.classification import LogisticRegression
6 from pyspark.ml.feature import VectorAssembler
7 from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit, CrossValidator
8 from pyspark.ml.evaluation import BinaryClassificationEvaluator
9 from pyspark.ml.classification import GBTCClassifier
10 from pyspark.ml.classification import DecisionTreeClassifier
11 from pyspark.ml.classification import RandomForestClassifier
12 from pyspark.ml.evaluation import MulticlassClassificationEvaluator
13
14
```

Below the code editor, a status bar indicates: "Command took 0.03 seconds -- by sburde@calstatela.edu at 5/18/2019, 10:04:20 PM on Ranger".

The second code block defines a DataFrame schema:

```
1 # DataFrame Schema, that should be a Table schema by Jongwook Woo (jwoo5@calstatela.edu) 01/07/2016
2 SeattleSchema = StructType([
3   StructField("BibNum", IntegerType(), False),
4   StructField("Title", StringType(), False),
5   StructField("Author", StringType(), False),
6   StructField("ISBN", StringType(), False),
7   StructField("PublicationYear", IntegerType(), False),
8   StructField("Publisher", StringType(), False),
9   StructField("ItemType", StringType(), False),
10  StructField("ItemCollection", StringType(), False),
11  StructField("FloatingItem", IntegerType(), False),
12  StructField("ItemLocation", StringType(), False),
13  StructField("ReportDate", StringType(), False),
14  StructField("ItemCount", IntegerType(), False),
15 ])
```

The bottom of the image shows a Windows taskbar with various application icons and a system clock indicating 10:15 AM on 5/19/2019.

## Step 3: Select data from the table

Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)

Question 1 (5%)

Read your flights.csv file from its table at Databricks

```
1 # TODO: use LibrarySchema in order to adopt shcmea to read csv data set in the schema.
2 # Jongwook Woo (jwoo5@calstatela.edu) 01/07/2016
3
4 # Load the source data
5 csv = spark.sql("SELECT * FROM seattle_library_optimized_file_cd44f_csv")
6
7 # csv: pyspark.sql.dataframe.DataFrame = [BibNum: integer, Title: string ... 11 more fields]
8 Command took 0.15 seconds -- by sburde@calstatela.edu at 5/18/2019, 10:04:20 PM on Ranger
```

```
1 # Select features and label
2 data = csv.select("BibNum", "PublicationYear", "ItemCount", col("FloatingItem").alias("label"))
3
4 # Split the data
5 splits = data.randomSplit([0.7, 0.3])
6 train = splits[0]
7 test = splits[1].withColumnRenamed("label", "trueLabel")
8
9 # data: pyspark.sql.dataframe.DataFrame = [BibNum: integer, PublicationYear: integer ... 2 more fields]
10 # train: pyspark.sql.dataframe.DataFrame = [BibNum: integer, PublicationYear: integer ... 2 more fields]
11 # test: pyspark.sql.dataframe.DataFrame = [BibNum: integer, PublicationYear: integer ... 2 more fields]
12 Command took 0.06 seconds -- by sburde@calstatela.edu at 5/18/2019, 10:05:51 PM on Ranger
```

Tutorial\_DATABR...doc

Show all

ENG 10:16 AM  
US 5/19/2019

## Step 4: Define for Train and Test the Data. (0.7 – Train, 0.3 – Test)

Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)

Attached: Ranger | File | View: Code | Permissions | Run All | Clear

2 # Jongwook Woo (jwoos5@calstatela.edu) 01/07/2016

3

4 # Load the source data

5 `csv = spark.sql("SELECT * FROM seattle_library_optimized_file_cd44f_csv")`

▶ `csv: pyspark.sql.dataframe.DataFrame = [BibNum: integer, Title: string ... 11 more fields]`

Command took 0.15 seconds -- by sburde@calstatela.edu at 5/18/2019, 10:04:28 PM on Ranger

Cnd 7

1 # Select features and label

2 `data = csv.select("BibNum", "PublicationYear", "ItemCount", col("FloatingItem").alias("label"))`

3

4 # Split the data

5 `splits = data.randomSplit([0.7, 0.3])`

6 `train = splits[0]`

7 `test = splits[1].withColumnRenamed("label", "trueLabel")`

▶ `data: pyspark.sql.dataframe.DataFrame = [BibNum: integer, PublicationYear: integer ... 2 more fields]`

▶ `train: pyspark.sql.dataframe.DataFrame = [BibNum: integer, PublicationYear: integer ... 2 more fields]`

▶ `test: pyspark.sql.dataframe.DataFrame = [BibNum: integer, PublicationYear: integer ... 2 more fields]`

Command took 0.06 seconds -- by sburde@calstatela.edu at 5/18/2019, 10:05:31 PM on Ranger

Cnd 8

**Define the Pipeline**

Now define a pipeline that creates a feature vector and trains a classification model

Cnd 9

Tutorial\_DATABR...doc

Show all

ENG 10:17 AM  
US 5/19/2019

## Step 5: Define Pipeline for the data

The screenshot shows a Databricks workspace interface. The notebook title is "Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)". The current step is "Define the Pipeline", which includes a command cell with the following Python code:

```
1 # Define the pipeline
2 assembler = []
3 lr = []
4 pipeline = []
5 for i in range(3):
6     assembler.insert(i, VectorAssembler(inputCols = ["BibNum", "PublicationYear", "ItemCount"], outputCol="features"))
7     lr.insert(i, LogisticRegression(labelCol="label", featuresCol="features"))
8     pipeline.insert(i, Pipeline(stages=[assembler[i], lr[i]]))
```

Below the code cell, there is a "Tune Parameters" section with text explaining how to use **TrainValidationSplit** to evaluate each combination of parameters defined in a **ParameterGrid** against a subset of the training data. Below that is a "Regularization" section.

## Step 5: Tune the parameter

### Tune Parameters

You can tune parameters to find the best model for your data. A simple way to do this is to use **TrainValidationSplit** to evaluate each combination of parameters defined in a **ParameterGrid** against a subset of the training data in order to find the best performing parameters.

### Regularization

is a way of avoiding Imbalances in the way that the data is trained against the training data so that the model ends up being over fit to the training data. In other words It works really well with the training data but it doesn't generalize well with other data. That we can use a **regularization parameter** to vary the way that the model balances that way.

### Training ratio of 0.7

it's going to use 70% of the the data that it's got in its training set to train the model and then the remaining 30% is going to use to validate the trained model.

In **ParamGridBuilder**, all possible combinations are generated from regParam, maxIter, threshold. So it is going to try each combination of the parameters with 70% of the data to train the model and 30% to validate it.

## Step 5a: Train Validation Split with Threshold parameters

The screenshot shows a Databricks notebook titled "Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)". The notebook is running on a cluster named "Noi". The interface includes a sidebar with navigation options like Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main area displays three code cells:

**Cell 13:** A comment indicating the first combination of parameters: (regParam: [0.01, 0.5]), (threshold: [0.30, 0.35]), (maxIter: [1, 5]).

```
1 # define list of models made from Train Validation Split and Cross Validation
2 model = []
```

**Cell 14:** Code to create a ParamGridBuilder and add a grid of parameters for regParam, threshold, and maxIter.

```
1 # params referred to the reference above
2 paramGrid = (ParamGridBuilder() \
3   .addGrid(lr[0].regParam, [0.01, 0.5, 2.0]) \
4   .addGrid(lr[0].threshold, [0.30, 0.35]) \
5   .addGrid(lr[0].maxIter, [1, 5]) \
6   .build())
```

**Cell 15:** Code to perform a TrainValidationSplit and insert the best model into the model list.

```
1 tvs = TrainValidationSplit(estimator=pipeline[0], evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid, trainRatio=0.8)
2 # the first best model
3 model.insert(0, tvs.fit(train))
```

The notebook also shows execution times for each cell: Cell 13 took 0.02 seconds, Cell 14 took 0.02 seconds, and Cell 15 took 1.01 minutes. The bottom of the screen shows a Windows taskbar with various application icons and a system clock indicating 10:24 AM on 5/19/2019.

## Step 5b: Train Validation Split with elastic-net parameters

The second combination of parameters with (regParam: [0.01, 0.5, 2.0]), (elasticNetParam: [0.0, 0.5, 1]), (maxIter: [1, 5])

The screenshot shows a Databricks notebook interface. The browser address bar displays a URL from community.cloud.databricks.com. The notebook title is "Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)". The left sidebar contains navigation icons for Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main content area shows two code cells. Cell 16, titled "Train Validation Split with elastic-net parameters", contains a "Question 2 (5%)" prompt asking to complete the second combination of parameters. Cell 17 contains a Python code snippet that defines a ParamGridBuilder and adds three parameter combinations for regParam, elasticNetParam, and maxIter. Cell 18 contains a Python code snippet that uses TrainValidationSplit to evaluate the parameter grid and insert the best model into the model list. The bottom of the notebook shows a status bar with the text "Tutorial\_DATABR...doc" and a "Show all" button. The Windows taskbar at the very bottom shows various application icons and the system clock indicating 10:25 AM on 5/19/2019.

```
1 # T000: params referred to the reference above
2 paramGrid2 = (ParamGridBuilder() \
3   .addGrid(lr[1].regParam, [0.01, 0.5, 2.0]) \
4   .addGrid(lr[1].elasticNetParam, [0.0, 0.5, 1]) \
5   .addGrid(lr[1].maxIter, [1, 5]) \
6   .build())

1 tvs2 = TrainValidationSplit(estimator=pipeLine[1], evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid2, trainRatio=0.8)
2
3 # the second best model
4 model.insert(1, tvs2.fit(train))
```



## Step 5c: Cross Validator with elastic net parameters

The combination of parameters with (regParam: [0.01, 0.5, 2.0]), (elasticNetParam: [0.0, 0.5, 1]), (maxIter: [1, 5])

The screenshot shows a Databricks notebook interface. The title bar reads "Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)". The notebook content includes a section titled "Cross Validator with elastic net parameters" with the instruction "Build the best model using Cross Validator". Below this is "Question 3 (5%)" which asks for the combination of parameters: (regParam: [0.01, 0.5, 2.0]), (elasticNetParam: [0.0, 0.5, 1]), (maxIter: [1, 5]).

Two code cells are visible:

```
Cmd 20
1 # TODO: params referred to the reference above
2 paramGridCV = (ParamGridBuilder() \
3               .addGrid(Lr[2].regParam, [0.01, 0.5, 2.0]) \
4               .addGrid(Lr[2].elasticNetParam, [0.0, 0.5, 1]) \
5               .addGrid(Lr[2].maxIter, [1, 5]) \
6               .build())

Command took 0.02 seconds -- by sburde@calstatela.edu at 5/18/2019, 10:08:51 PM on Ranger
```

```
Cmd 21
1 # TODO: K = 2 you may test it with 5, 10
2 # K=2, 3, 5,
3 # K= 10 takes too long
4 cv = CrossValidator(estimator=pipeline[2], evaluator=BinaryClassificationEvaluator(), \
5                   estimatorParamMaps=paramGridCV, numFolds=5)
6
7 # the third best model
8 model_index = cv.bestModel.index
```

The bottom of the image shows a Windows taskbar with various application icons and a system clock indicating 10:30 AM on 5/19/2019.

Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)

Attached: Ranger | File | View Code | Permissions | Run All | Clear

```
1 # TODO: params referred to the reference above
2 paramGridCV = (ParamGridBuilder() \
3   .addGrid(Lr[2].regParam, [0.01, 0.5, 2.0]) \
4   .addGrid(Lr[2].elasticNetParam, [0.0, 0.5, 1]) \
5   .addGrid(Lr[2].maxIter, [1, 5]) \
6   .build())
```

Command took 0.02 seconds -- by sburde@calstatela.edu at 5/18/2019, 10:08:51 PM on Ranger

Cmd 21

```
1 # TODO: K = 2 you may test it with 5, 10
2 # K=2, 3, 5,
3 # K= 10 takes too long
4 cv = CrossValidator(estimator=pipeline[2], evaluator=BinaryClassificationEvaluator(), \
5   estimatorParamMaps=paramGridCV, numFolds=5)
6
7 # the third best model
8 model.insert(2, cv.fit(train))
```

▶ (48) Spark Jobs

Command took 4.00 minutes -- by sburde@calstatela.edu at 5/18/2019, 10:08:56 PM on Ranger

Cmd 22

### Test the Model

Now you're ready to apply the model to the test data.

Cmd 23

Tutorial\_DATABRI...doc

Show all X

Windows taskbar: 10:31 AM, 5/19/2019

## Step 6: Test The model

Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)

Now you're ready to apply the model to the test data.

```
1 # list prediction
2 prediction = []
3 predicted = []
4 for i in range(3):
5     prediction.insert(i, model[i].transform(test))
6     predicted.insert(i, prediction[i].select("features", "prediction", "probability", "trueLabel"))
7     predicted[i].show(30)
```

(3) Spark Jobs

features	prediction	probability	trueLabel
[444.0, 1969.0, 1.0]	0.0	[0.86075529466693...	0
[922.0, 1968.0, 1.0]	0.0	[0.86074758840554...	0
[1508.0, 1968.0, 1.0]	0.0	[0.86074599741710...	0
[2108.0, 1969.0, 1.0]	0.0	[0.86074824276446...	0
[2157.0, 1969.0, 1.0]	0.0	[0.86074801322191...	0
[2763.0, 1969.0, 1.0]	0.0	[0.86074543722273...	0
[3281.0, 1970.0, 3.0]	0.0	[0.85949325246698...	0
[4920.0, 1971.0, 2.0]	0.0	[0.86012006363403...	0
[6470.0, 1970.0, 1.0]	0.0	[0.86073535362141...	0
[7434.0, 1971.0, 1.0]	0.0	[0.86073693051421...	0

## Step 7: Compute Confusion Matrix Metrics

Classifiers are typically evaluated by creating a confusion matrix, which indicates the number of:

- True Positives
- True Negatives
- False Positives
- False Negatives

From these core measures, other evaluation metrics such as precision and recall can be calculated.

databricks

Home

Workspace

Recents

Data

Clusters

Jobs

Search

Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)

Attached: Ranger File View: Code Permissions Run All Clear Publish Comments Revision history

Cmd 26

```
1 # TODO: Complete the following [Fill-In] to calculate Recall
2 for i in range(2):
3     tp = float(predicted[i].filter("prediction == 1.0 AND truelabel == 1").count())
4     fp = float(predicted[i].filter("prediction == 1.0 AND truelabel == 0").count())
5     tn = float(predicted[i].filter("prediction == 0.0 AND truelabel == 0").count())
6     fn = float(predicted[i].filter("prediction == 0.0 AND truelabel == 1").count())
7     metrics = spark.createDataFrame([
8         ("TP", tp),
9         ("FP", fp),
10        ("TN", tn),
11        ("FN", fn),
12        ("Precision", tp / (tp + fp)),
13        ("Recall", tp / (tp + fn)), ["metric", "value"])
14    metrics.show()
```

▶ (4) Spark Jobs

ZeroDivisionError: float division by zero

Command took 5.12 seconds -- by sburde@calstatela.edu at 5/19/2019, 10:14:35 PM on Ranger

Cmd 27

### Review the Area Under ROC

Another way to assess the performance of a classification model is to measure the area under a ROC curve for the model. the spark.ml library includes a **BinaryClassificationEvaluator** class that you can use to compute this.

Cmd 28

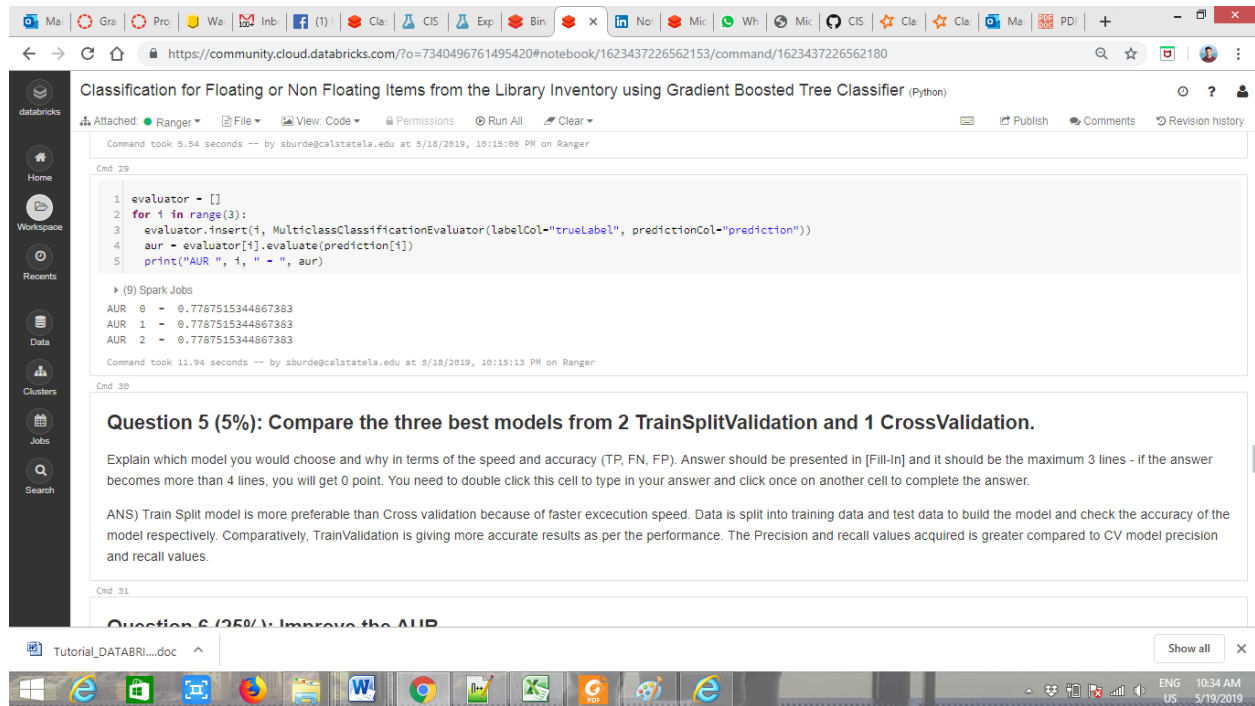
Tutorial\_DATABRI...doc

Show all X

Windows Taskbar

System Tray

## Step 8: Review the Area Under ROC



Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)

Command took 5.54 seconds -- by sburde@calstatela.edu at 5/18/2019, 10:15:08 PM on Ranger

```
1 evaluator = []
2 for i in range(3):
3     evaluator.insert(i, MulticlassClassificationEvaluator(labelCol="trueLabel", predictionCol="prediction"))
4     aur = evaluator[i].evaluate(prediction[i])
5     print("AUC ", i, " = ", aur)
```

▶ (9) Spark Jobs

AUC	Model	Value
0	Train Split	0.7787515344867383
1	Train Validation Split	0.7787515344867383
2	Cross Validation	0.7787515344867383

Command took 11.94 seconds -- by sburde@calstatela.edu at 5/18/2019, 10:15:13 PM on Ranger

**Question 5 (5%): Compare the three best models from 2 TrainSplitValidation and 1 CrossValidation.**

Explain which model you would choose and why in terms of the speed and accuracy (TP, FN, FP). Answer should be presented in [Fill-In] and it should be the maximum 3 lines - if the answer becomes more than 4 lines, you will get 0 point. You need to double click this cell to type in your answer and click once on another cell to complete the answer.

ANS) Train Split model is more preferable than Cross validation because of faster execution speed. Data is split into training data and test data to build the model and check the accuracy of the model respectively. Comparatively, TrainValidation is giving more accurate results as per the performance. The Precision and recall values acquired is greater compared to CV model precision and recall values.

**Question 6 (25%): Improve the AUC**

## Step 9: Improve AUC: Gradient Boosted Tree Classifier

Gradient boosted tree classifier model using Multiclass Classification evaluator help us to increase the accuracy (AUC). I am using training split 0.8 with TrainValidationSplit model

## Step 9.a Define assembler and pipeline for GBT

The screenshot shows a Databricks notebook interface. The title bar reads "Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)". The notebook content includes a text block explaining the goal of using a Gradient Boosted Tree classifier with a Multiclass Classification evaluator to increase accuracy (AUR) using a training split of 0.8 with a TrainValidationSplit model. Below this, there are three code blocks. The first code block (Cmd 33) imports Spark SQL and Spark ML libraries, and the GBTClassifier. The second code block (Cmd 34) is titled "Define assembler and pipeline for GBT" and contains Python code to define a VectorAssembler, a GBTClassifier, and a Pipeline. The third code block (Cmd 35) is titled "Train Validation Split with Bins parameters" and contains text describing the combination of parameters: (maxDepth: [2, 4, 6]), (maxBins: [20, 60]), (maxIter: [10, 20]). The notebook interface also shows a sidebar with navigation options like Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The bottom of the screen shows a Windows taskbar with various application icons and a system clock indicating 10:37 AM on 5/19/2019.

Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)

Gradient boosted tree classifier model using Multiclass Classification evaluator help us to increase the accuracy (AUR). I am using training split 0.8 with TrainValidationSplit model

Cmd 33

```
1 # Import Spark SQL and Spark ML library
2 from pyspark.ml.classification import GBTClassifier
```

Command took 0.02 seconds -- by sburde@calstatela.edu at 5/18/2019, 10:15:37 PM on Ranger

Cmd 34

**Define assembler and pipeline for GBT**

```
1 #Define assembler and pipeline for GBT
2 assembler_GBT = VectorAssembler(inputCols = ["BibNum", "PublicationYear", "ItemCount"], outputCol="features")
3 gbt = GBTClassifier(labelCol="label", featuresCol="features", maxIter=10)
4 pipeline = Pipeline(stages=[assembler_GBT, gbt])
```

Command took 0.06 seconds -- by sburde@calstatela.edu at 5/18/2019, 10:16:25 PM on Ranger

Cmd 35

**Train Validation Split with Bins parameters**

The combination of parameters with (maxDepth: [2, 4, 6]), (maxBins: [20, 60]), (maxIter: [10, 20])

Cmd 37

## Step 9.b Train Validation Split with Bins parameters

Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)

Attached: Ranger | File | View: Code | Permissions | Run All | Clear

Cnd 37

```
1 #Define paramGrid for GBT
2 paramGrid_GBT = (ParamGridBuilder() \
3     .addGrid(gbt.maxDepth, [2, 4, 6]) \
4     .addGrid(gbt.maxBins, [28, 68]) \
5     .addGrid(gbt.maxIter, [18, 28]) \
6     .build())
```

Command took 0.03 seconds -- by sburde@calstatela.edu at 5/18/2019, 10:16:32 PM on Ranger

Cnd 38

### Test the Model

Now you're ready to apply the model to the test data.

Cnd 39

```
1 #TrainValidation split for BinaryClassification
2 gbt_tvs = TrainValidationSplit(estimator=pipeline, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid_GBT, trainRatio=0.8)
3 model_gbt = gbt_tvs.fit(train)
```

► (56) Spark Jobs

Command took 5.28 minutes -- by sburde@calstatela.edu at 5/18/2019, 10:16:38 PM on Ranger

Cnd 40

### Transform the data with Testing Splits

Tutorial\_DATABR...doc | Show all | X

10:37 AM 5/19/2019

## Step 9.b Test the Model

The screenshot shows a Databricks notebook interface. The title bar reads "Classification for Floating or Non Floating Items from the Library Inventory using Gradient Boosted Tree Classifier (Python)". The left sidebar contains navigation icons for Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main area displays two code blocks. The first block, titled "Test the Model", contains a single command (Cmd 39) with Python code for training a Gradient Boosted Tree classifier. The second block, titled "Transform the data with Testing Splits", contains a single command (Cmd 40) with Python code for applying the trained model to test data and showing the results. The bottom of the screen shows a Windows taskbar with various application icons and a system clock indicating 10:38 AM on 5/19/2019.

```
1 #TrainValidation split for BinaryClassification
2 GBT_tvs = TrainValidationSplit(estimator=GradientBoostedTreeClassifier(), evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid_GBT, trainRatio=0.8)
3 model_gbt = GBT_tvs.fit(train)
```

```
1 #Apply Transform to testing split and show 30 rows
2 prediction_gbt = model_gbt.transform(test)
3 predicted_gbt = prediction_gbt.select("features", "prediction", "probability", "trueLabel")
4 predicted_gbt.show(30)
```

## Step 9. c Review the Area under ROC

The screenshot shows the same Databricks notebook interface, now displaying the third code block titled "Review the Area Under ROC". This block (Cmd 42) contains a text explanation of the Area Under the ROC curve (AUR) and its importance in evaluating classification models. Below this, a single command (Cmd 43) shows Python code to calculate the AUR using the MulticlassClassificationEvaluator class. The output of the command is displayed, showing an AUR value of approximately 0.787. The final part of the screenshot shows the start of a new section titled "Result", which explains that the Multiclass Classification Evaluator is more accurate than the Binary Class Classifier. The bottom of the screen shows the same Windows taskbar as the previous screenshot.

```
1 #print the AUR from the GBT model using Multiclass Classifier
2 evaluator_gbt = MulticlassClassificationEvaluator(LabelCol="trueLabel", predictionCol="prediction")
3 aur_gbt = evaluator_gbt.evaluate(prediction_gbt)
4 print("AUR = ", aur_gbt)
```

AUR = 0.7873476526714038

### Result

Area under ROC is much more accurate when using Multiclass Classification Evaluator than Binary Class Classifier using Gradient Boost Tree Classifier



# Result

Area under ROC is much more accurate when using Multiclass Classification Evaluator than Binary Class Classifier using Gradient Boost Tree Classifier is 0.787.

## Step 10. Random Forest Classifier

Random Forest classifier model using Multiclass Classification evaluator help us to increase the accuracy (AUR). I am using training split 0.8 with TrainValidationSplit model

## Define assembler and pipline for RFC

The screenshot displays a Databricks workspace interface. The top navigation bar includes options like 'Detached', 'File', 'View Code', 'Permissions', 'Run All', and 'Clear'. The main content area is titled 'Classification for Floating or Non Floating Items from the Library Inventory using Random Forest Classifier (Python)'. It shows a sequence of commands in a notebook:

- Cmd 34:** A command box with the title 'Define assembler and pipline for RFC'. The code defines a `VectorAssembler` for features and a `RandomForestClassifier` with `numTrees=10`, then combines them into a `Pipeline`.
- Cmd 36:** A command box with the title 'Train Validation Split with Bins parameters'. It contains a text description: 'The combination of parameters with (maxDepth: [2, 4, 6]), (maxBins: [20, 60]), (maxIter: [10, 20])'.
- Cmd 37:** A command box with the title 'Define paramGrid for GBT'. The code uses `ParamGridBuilder` to create a grid for `maxDepth`, `maxBins`, and `numTrees`.

The bottom of the image shows a Windows taskbar with various application icons and a system clock indicating 10:43 AM on 5/19/2019.

## Train Validation Split with Bins parameters

Classification for Floating or Non Floating Items from the Library Inventory using Random Forest Classifier (Python)

The combination of parameters with (maxDepth: [2, 4, 6]), (maxBins: [20, 60]), (maxIter: [10, 20])

```
1 #Define paramGrid for GBT
2 paramGrid_RFC = (ParamGridBuilder() \
3   .addGrid(rfc.maxDepth, [2, 4, 6]) \
4   .addGrid(rfc.maxBins, [20, 60]) \
5   .addGrid(rfc.numTrees, [10, 20]) \
6   .build())
```

Command took 0.02 seconds -- by sburde@calstatela.edu at 5/19/2019, 12:11:48 AM on Ranger

### Test the Model

Now you're ready to apply the model to the test data.

```
1 #TrainValidation split for BinaryClassification
2 RFC_tvs = TrainValidationSplit(estimator=pipeLine, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid_RFC, trainRatio=0.8)
3 model_rfc = RFC_tvs.fit(train)
```

(57) Spark Jobs

Command took 44.98 seconds -- by sburde@calstatela.edu at 5/19/2019, 12:12:52 AM on Ranger

## Test the Model

Classification for Floating or Non Floating Items from the Library Inventory using Random Forest Classifier (Python)

Now you're ready to apply the model to the test data.

```
1 #TrainValidation split for BinaryClassification
2 RFC_tvs = TrainValidationSplit(estimator=pipeLine, evaluator=BinaryClassificationEvaluator(), estimatorParamMaps=paramGrid_RFC, trainRatio=0.8)
3 model_rfc = RFC_tvs.fit(train)
```

(57) Spark Jobs

Command took 44.98 seconds -- by sburde@calstatela.edu at 5/19/2019, 12:12:52 AM on Ranger

### Transform the data with Testing Splits

```
1 #Apply Transform to testing split and show 30 rows
2 prediction_rfc = model_rfc.transform(test)
3 predicted_rfc = prediction_rfc.select("features", "prediction", "probability", "trueLabel")
4 predicted_rfc.show(30)
```

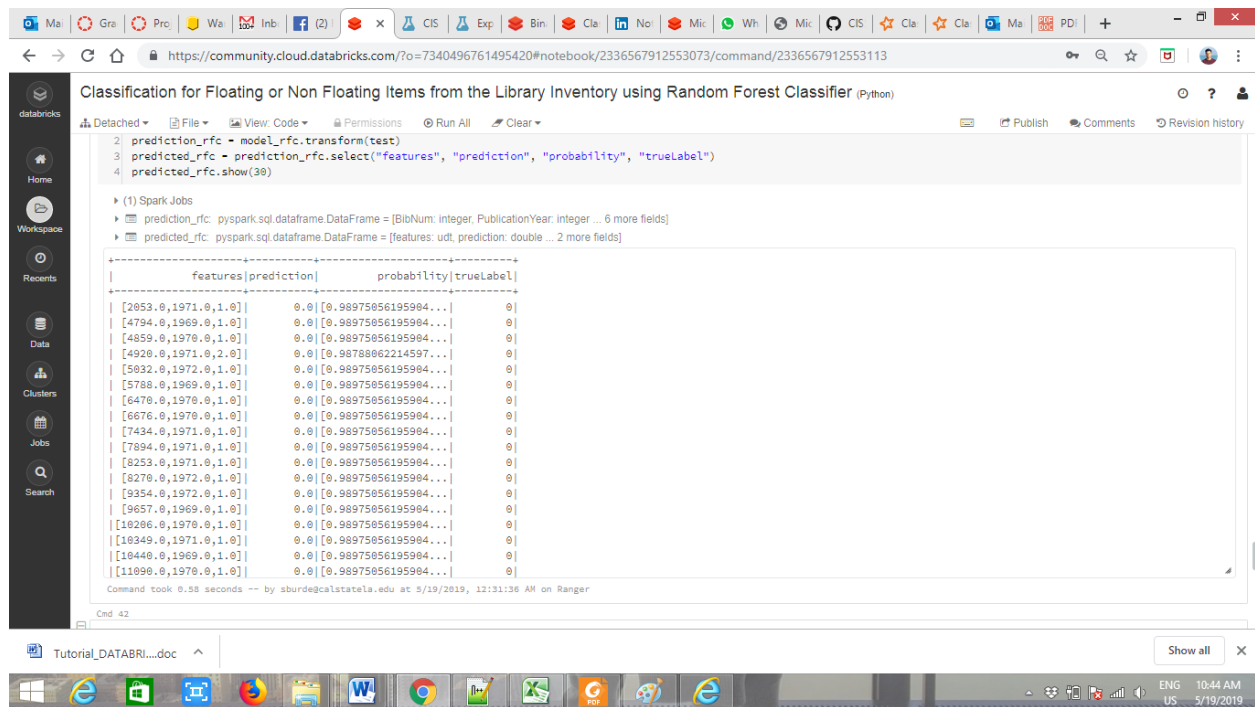
(1) Spark Jobs

prediction\_rfc: pyspark.sql.dataframe.DataFrame = [BitNum: integer, PublicationYear: integer ... 6 more fields]

predicted\_rfc: pyspark.sql.dataframe.DataFrame = [features: udt, prediction: double ... 2 more fields]

features	prediction	probability	trueLabel
[2053.0,1971.0,1.0]	0.0	[0.98975856195984...	0
[4794.0,1969.0,1.0]	0.0	[0.98975856195984...	0
[4859.0,1978.0,1.0]	0.0	[0.98975856195984...	0

## Transform the data with Testing Splits



Classification for Floating or Non Floating Items from the Library Inventory using Random Forest Classifier (Python)

```
2 prediction_rfc = model_rfc.transform(test)
3 predicted_rfc = prediction_rfc.select("features", "prediction", "probability", "trueLabel")
4 predicted_rfc.show(30)
```

(1) Spark Jobs

- prediction\_rfc: pyspark.sql.dataframe.DataFrame = [BitNum: integer, PublicationYear: integer ... 6 more fields]
- predicted\_rfc: pyspark.sql.dataframe.DataFrame = [features: udt, prediction: double ... 2 more fields]

features	prediction	probability	trueLabel
[2052.0,1971.0,1.0]	0.0	[0.98975856195904...	0
[4794.0,1969.0,1.0]	0.0	[0.98975856195904...	0
[4859.0,1970.0,1.0]	0.0	[0.98975856195904...	0
[4920.0,1971.0,2.0]	0.0	[0.98788662214597...	0
[5832.0,1972.0,1.0]	0.0	[0.98975856195904...	0
[5788.0,1969.0,1.0]	0.0	[0.98975856195904...	0
[6470.0,1970.0,1.0]	0.0	[0.98975856195904...	0
[6676.0,1970.0,1.0]	0.0	[0.98975856195904...	0
[7434.0,1971.0,1.0]	0.0	[0.98975856195904...	0
[7894.0,1971.0,1.0]	0.0	[0.98975856195904...	0
[8253.0,1971.0,1.0]	0.0	[0.98975856195904...	0
[8270.0,1972.0,1.0]	0.0	[0.98975856195904...	0
[9354.0,1972.0,1.0]	0.0	[0.98975856195904...	0
[9657.0,1969.0,1.0]	0.0	[0.98975856195904...	0
[10286.0,1970.0,1.0]	0.0	[0.98975856195904...	0
[10349.0,1971.0,1.0]	0.0	[0.98975856195904...	0
[10440.0,1969.0,1.0]	0.0	[0.98975856195904...	0
[11090.0,1970.0,1.0]	0.0	[0.98975856195904...	0

Command took 0.58 seconds -- by sburde@calstatela.edu at 5/19/2019, 12:31:36 AM on Ranger

## Review the Area Under ROC by Random Forest Classifier

The screenshot displays a Databricks notebook titled "Classification for Floating or Non Floating Items from the Library Inventory using Random Forest Classifier (Python)". The notebook is in a "Detached" state. The main content area shows a section titled "Review the Area Under ROC" with the text: "The performance of a classification model is to measure the area under a ROC curve using MulticlassClassificationEvaluator class to gain more accuracy compared with other classifier". Below this, a code cell (Cmd 43) contains the following Python code:

```
1: #print the AUR from the GBT model using Multiclass Classifier
2: evaluator_rfc = MulticlassClassificationEvaluator(LabelCol="trueLabel", predictionCol="prediction")
3: aur_rfc = evaluator_rfc.evaluate(prediction_rfc)
4: print("AUR - ", aur_rfc)
```

The output of the code cell shows the AUR value: "AUR - 0.7880385826868218". The command took 2.77 seconds to execute. Below the code cell, a section titled "Result" states: "Area under ROC is much more accurate when using Multiclass Classification Evaluator than Binary Class Classifier using Gradient Boost Tree Classifier". A final text block (Cmd 45) provides a reference to improve accuracy: "You may build an experiment in Azure ML Studio to get hints to add the function of data engineering or google to find out some hints how others have done. The following is the reference that you may look at as well".

## Result

Area under ROC is much more accurate when using Multiclass Classification Evaluator than Binary Class Classifier using Random Forest Classifier is 0.788.

## References to improve the accuracy

1. Extracting, transforming and selecting features, <https://spark.apache.org/docs/latest/ml-features.html>
2. Basic data preparation in Pyspark — Capping, Normalizing and Scaling, <http://bit.ly/2lhs6Wa>
3. Machine Learning with PySpark and MLlib — Solving a Binary Classification Problem, <http://bit.ly/2Zb20tg>