# [https://avatars2.githubusercontent.com/u/4156894?v=3&s=100](http://www.calstatela.edu/centers/hipic) CIS5560 Term Project Tutorial

#### Authors: Jeffrey Dang; Grace Mayorga

#### Instructor: [Jongwook Woo](https://www.linkedin.com/in/jongwook-woo-7081a85)

#### Date: 05/18/2022

**Lab Tutorial**

Jeffrey Dang ([jdang8@calstatela.edu](mailto:jdang8@calstatela.edu))

Grace Mayorga ([gnunez8@calstatela.edu](mailto:gnunez8@calstatela.edu))

05/21/2022

**Employment Wages in Education versus the County​**

**Objectives**

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

* How to download the dataset from GitHub
* Create Spark cluster
* Use Machine Learning Algorithms
  + Linear Regression
  + Random Forest Regression
  + Gradient Boot Tree Regression
    - Will use CrossValidation and TrainValidation to train the model
* Measuring Accuracy with R2 and RMSE

**Platform Spec**

* **Databricks Runtime version:** 10.4 LTS (includes Apache Spark 3.2.1, Scala 2.12)
* **Memory:** 15.3 GB , 2 cores , 1 DBU, Nodes : 1
* **File System:** DBFS (Data Bricks File System)
* **Python Version** 3.8.10

Step 1: Creating a Cluster

1. Sign into your Databricks account at <https://community.cloud.databricks.com/login.html>
2. On the left hand side, click Create -> Cluster  
   Graphical user interface, application

   Description automatically generated
3. Enter a name for your cluster and click “Create Cluster”  
   Graphical user interface, text, application

   Description automatically generated

Step 2: Getting the dataset

1. Go to <https://csula.sharepoint.com/sites/CIS5560Project62/Shared%20Documents/Forms/AllItems.aspx?id=%2Fsites%2FCIS5560Project62%2FShared%20Documents%2FGeneral%2FProject%20Submission%2FDataset&p=true&ga=1> and download the dataset that will be used in this project. The data set has 30 csv files.
2. Once you have successfully downloaded the dataset, go to your data bricks account and select Data -> Create Table  
   Graphical user interface, text, application, chat or text message

   Description automatically generated
3. Enter the name “Project” within the DBFS Target Directory and upload the dataset.

Graphical user interface, text, application, email

Description automatically generated

Step 3: Creating a Notebook

1. Click on Workspace
2. Click on “Users”
3. Select your User
4. Right click within the blank space and select Create-> Notebook

Graphical user interface, text, application

Description automatically generated

Step 4: Setting up the Notebook

1. Enter a name for your Notebook
2. Select the active cluster you had created in step 1

Step 5: Import PySpark SQL Libraries

Import all the Spark SQL and ML libraries as mentioned below. This is necessary to access the functions available in those libraries. Run the code listed below.

from pyspark.sql.types import \*

from pyspark.sql.functions import \*

from pyspark.ml import \*

from pyspark.ml.regression import LinearRegression , DecisionTreeRegressor, RandomForestRegressor, GBTRegressionModel, GBTRegressor

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit, CrossValidator

from pyspark.ml.evaluation import RegressionEvaluator

from pyspark.ml.feature import VectorIndexer

from pyspark.ml.feature import VectorAssembler, MinMaxScaler

from pyspark.ml.linalg import Vectors

from pyspark.ml.evaluation import BinaryClassificationEvaluator

from pyspark.sql import SparkSession

from pyspark.sql import functions as F

from pyspark.context import SparkContext

from pyspark.mllib.evaluation import MulticlassMetrics

from pyspark.sql.session import SparkSession

Step 5: Referencing the csv file from DBFS (Databricks File System)

Here we are using the variable name "path" to reference all the files within our Project folder. After we have used the variable df to be able to call the dataset with the applied options. Run the code listed below.

path = ['/FileStore/tables/Project/\*.csv']

file\_type = "csv"

# CSV options

infer\_schema = "true"

first\_row\_is\_header = "true"

delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.

df = spark.read.format(file\_type).option("inferSchema", infer\_schema)\

.option("header", first\_row\_is\_header) \

.option("sep", delimiter) \

.load(path)

df.show()

Table

Description automatically generated with low confidence

Step 6: Display your Dataset

Displaying the dataset to ensure that it was imported successfully. Run the code listed below

print("Whole Data Set")

display(df)  
Graphical user interface, table

Description automatically generated

Step 7: Cleaning the dataset

Create a temporary view of the dataframe 'df' and seeing what columns would be good for this learning algorithm. Run the code listed below

# Create a view or table

temp\_table\_name = "masterTable"

df.createOrReplaceTempView(temp\_table\_name)

dftemp=df.select('Year','EmployerName','Position','TotalWages', 'RegularPay', 'MaxPositionSalary', 'MinPositionSalary')

dftemp.printSchema()

Text, letter

Description automatically generated

Step 8: Dropping the columns that will not be needed

Creating a new variable called df2 which will hold the dataset with the dropped columns. Run the code listed below.

df2 = df.drop("DepartmentOrSubdivision", "ElectedOfficial", "Judicial","OtherPositions", "DefinedBenefitPlanContribution","EmployeesRetirementCostCovered", "DeferredCompensationPlan","HealthDentalVision", "TotalRetirementAndHealthContribution","PensionFormula", "EmployerURL","EmployerPopulation","IncludesUnfundedLiability", "SpecialDistrictActivities"," IncludesUnfundedLiability","SpecialDistrictType","ReportedBaseWage","LumpSumPay","OtherPay","LastUpdatedDate")

#df2.printSchema()

print("Dataset with Dropped Columns")

df2.show()

Graphical user interface, table

Description automatically generated

Step 9: Filling in the null values with 0

We will be cleaning the dataset by having the null values with 0 and creating the new table view with the assigned variable of df3. Run the code listed below.

df3 = df2.na.fill(value=0).na.fill("NA")

print("Dataset with Dropped Columns")

df3.show()

Table

Description automatically generated with medium confidence

Step 10: Split the Data

In order to train the data, we had decided to do a split of 70 and 30. Run the code below:

splits = df3.randomSplit([0.7,0.3])

train = splits[0]

test = splits[1]

train\_rows = train.count()

test\_rows = test.count()

print("Training Rows:", train\_rows, "Testing Rows:", test\_rows)



Step 11: Define the Pipeline

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

assembler = VectorAssembler(inputCols = ["RegularPay", "TotalWages","MaxPositionSalary", "MinPositionSalary"], outputCol="wages")

Step 12: Training the model

We will use the dataframe, assembler, with the train dataframe listed above. The results will then be the new variable introduced as "wages." Run the code listed below.

training = assembler.transform(train)

print("Displaying results from Training the dataset")

training.show()

Table

Description automatically generated

Step 13: Using the Random Forest Regression to find Root Mean Square Error (RMSE) and R2.

1. RMSE Measures the difference between predicted values and measured values. Begin by creating the variable rf to use within your regression model with defining the amount of trees and iterations. Run the code listed below.

rf = RandomForestRegressor(labelCol="TotalWages", featuresCol="wages", numTrees=10, maxDepth=5)

# Combine stages into pipeline

pipeline = Pipeline(stages=[assembler, rf])

model = pipeline.fit(train)

1. With the model variable that was created by using the pipeline and trained dataframe, we will begin to find R2 and RMSE. We will be calling the test variable which was defined in cmd 18, which carries the value of "test = splits[1]" and run it against our model. Within the new variable predicted, we are able to display our findings with the new column TotalWages. Run the code listed below.

prediction = model.transform(test)

predicted = prediction.select("wages", "prediction", "TotalWages")

print("Displaying results from the Prediction")

predicted.show()

Table

Description automatically generated

1. Using the regression evaluator against our prediction data created above, we will be able to to find r2 and RMSE. Run the code below.

rf\_evaluator = RegressionEvaluator(predictionCol="prediction", \

labelCol="TotalWages",metricName="r2")

print("R Squared (R2) on test data = %g" % rf\_evaluator.evaluate(prediction))

rf\_evaluator = RegressionEvaluator(labelCol="TotalWages", predictionCol="prediction", metricName="rmse")

print("RMSE: %f" % rf\_evaluator.evaluate(prediction))

Text

Description automatically generated with low confidence

Step 14: Cleaning our Variable to be reused

Here we will be recalling the variables so they can be repurposed after being used in the Random Forest regression model.

1. By calling the df2 variable and getting rid of the null values, df3 will be able to be used for our next model, Gradient Boost Tree Regression. Run the code below.

df3 = df2.na.fill(value=0).na.fill("NA")

1. With the trained data being used before, we had repurposed the variables train and test to be called again. For our GBT regression model, we had done the same split as before, which is 70:30. Run the code listed below.

splits = df3.randomSplit([0.7,0.3])

train = splits[0]

test = splits[1]

train\_rows = train.count()

test\_rows = test.count()

print("Training Rows:", train\_rows, "Testing Rows:", test\_rows)



1. With the dataset being the same, we had prepared the assembler that will be called later. As before, we will be using the columns; RegularPay, TotalWages,MaxPositionSalary, and MinPositionSalary with the output data to be under "wages". Run the code listed below.

assembler = VectorAssembler(inputCols = ["RegularPay", "TotalWages","MaxPositionSalary", "MinPositionSalary"], outputCol="wages")

1. Now we will be using the train variable that was created and run that dataframe with the assembler. Run the code listed below.

training = assembler.transform(train)

Step 15: Using Gradient Boost Tree Regression to find R2 and RMSE

CrossValidator begins by splitting the dataset into a set of 2 folds that will be used as separate training and test datasets. By using CrossValidator to evaluate each set of parameters defined, we will be able to find the best model for our data.

1. With using the ParameterGrid against multiple folds of the data split into training and validation dataset. Run the code listed below.

gbt = GBTRegressor(labelCol="TotalWages", featuresCol="wages", maxDepth=5)

paramGrid = ParamGridBuilder()\

.addGrid(gbt.maxDepth, [2, 5])\

.addGrid(gbt.maxIter, [10, 20])\

.build()

gbt\_evaluator = RegressionEvaluator(predictionCol="prediction", labelCol="TotalWages", metricName="r2")

1. Create a pipeline variable with and using the parameters created above with CrossValidator. Run the code listed below.

pipeline = Pipeline(stages=[assembler, gbt])

cv = CrossValidator(estimator=pipeline, evaluator= gbt\_evaluator, estimatorParamMaps=paramGrid, numFolds=2)

model = cv.fit(train)

1. After we have ran our dataframe against the CrossValidator class, we will train the model to determine the predction results along with the results of R2 and rmse. Run the code listed below.

prediction = model.transform(test)

predicted = prediction.select("wages", "prediction", "TotalWages")

print("Displaying results from CrossValidator with the Predicted dataset")

predicted.show()

Graphical user interface, application, table

Description automatically generated

1. Run the code below to find the results for r2 and RMSE.

print("R Squared (R2) on test data = %g" % gbt\_evaluator.evaluate(prediction))

gbt\_evaluator = RegressionEvaluator(labelCol="TotalWages", predictionCol="prediction", metricName="rmse")

print("RMSE: %f" % gbt\_evaluator.evaluate(prediction))

Text

Description automatically generated with medium confidence

1. TrainValidationSplit is another method for hyper-parameter tuning. TrainValidationSplit will only evaluates each combination of parameters once, as opposed to a set amount specified like CrossValidator. Run the code below.

tv = TrainValidationSplit(estimator= pipeline, evaluator=gbt\_evaluator, estimatorParamMaps=paramGrid, trainRatio=0.8)

results = tv.fit(train)

# Transform the test data and generate predictions by applying the trained model

prediction = results.transform(test)

predicted = prediction.select("wages", "prediction", "TotalWages")

print("Displaying results from TrainValidationSplit with the Predicted dataset")

predicted.show()

Table

Description automatically generated

References

* + URL of Data Source, <https://csula.sharepoint.com/:f:/s/CIS5560Project62/Ehbr19c16rJHsATW_mqwA1kBTlXWoP1KOSKoNBEVU77BjQ?e=h8aHdc>
  + URL of your Github, <https://github.com/gnunez8/CIS5560-Project>