Checkpoint 4: Machine Learning **GROUP 5**
MSAI 339 **Jack R**
Prof. Jennie Rogers **Quincia H**
November 13, 2018 **Ikhlas A**

Purpose

The purpose of this checkpoint is to apply classifiers and regression algorithms from Machine Learning to the clean data we have been accumulating over the past few checkpoints. Using a spark cluster, we are able to severely reduce computation time for the algorithms.

Questions From Proposal

The questions we derived for the proposal did not fully encompass the knowledge we gained over the past few checkpoints. To successfully build off the previous checkpoints, we felt it was necessary to rethink the questions we were attempting answer or predict using a machine learning model.

How reasonable is a complaint, given the history of the officer and the area it was filed in?

The initial reason for this prediction is that complaints can often be emotionally-driven, more so than objectively. That being said, the labeling of any data would be strictly speculative at that point, as there is no concrete scale for reason.

For a given district, can we predict the number of complaints during an upcoming time period?

Our initial thought was that if a crime type's rate dramatically changes over a specific time period, it would be helpful for local residents to know in advance. That being said, this would require building 23 different models for each district based on how we stated the question. Also, it ended up being a bit brought of a question that would require a lot of accumulated datasets and features to provide anything fruitful.

Machine Learning Predictions

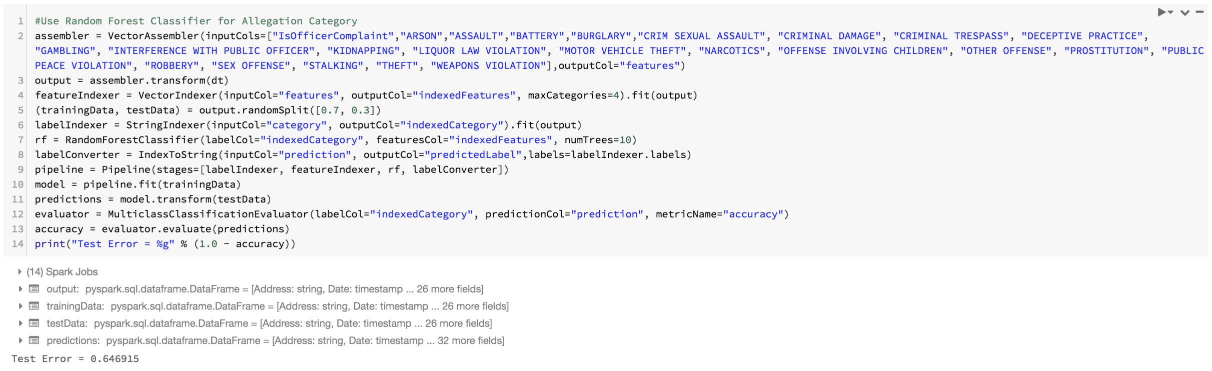
The predictions we pursued came off as more beneficial than the questions described in the proposal. The overarching theme of the two questions we attempted to follow was to try to predict something in one database almost strictly from another database, and vice-versa. This would show a convincing correlation.

Allegation Category

The first prediction we implemented was to predict the Allegation category based off the crimes that were committed on the given date and location of an allegation. A successful model could prove that specific allegations come hand in hand with certain types of crimes, especially in locations with bystanders.

Code

To prepare the data, we took the allegation to crime data table we had from the previous checkpoints and one-hot-encoded the CrimeType of each crime. After this, we added the binary strings up that corresponded to the same allegation (as there could be multiple crimes in a given location and date). Using the strengths of RandomForestClassifiers, we built a model.



```
#Use Random Forest Classifier for Allegation Category
assembler = VectorAssembler(inputCols=["IsOfficerComplaint","ARSON","ASSAULT","BATTERY","BURGLARY","CRIM SEXUAL ASSAULT", "CRIMINAL DAMAGE", "CRIMINAL TRESPASS", "DECEPTIVE PRACTICE", "GAMBLING", "INTERFERENCE WITH PUBLIC OFFICER", "KIDNAPPING", "LIQUOR LAW VIOLATION", "MOTOR VEHICLE THEFT", "NARCOTICS", "OFFENSE INVOLVING CHILDREN", "OTHER OFFENSE", "PROSTITUTION", "PUBLIC PEACE VIOLATION", "ROBBERY", "SEX OFFENSE", "STALKING", "THEFT", "WEAPONS VIOLATION"],outputCol="features")
output = assembler.transform(dt)
featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(output)
(trainingData, testData) = output.randomSplit([0.7, 0.3])
labelIndexer = StringIndexer(inputCol="category", outputCol="indexedCategory").fit(output)
rf = RandomForestClassifier(labelCol="indexedCategory", featuresCol="indexedFeatures", numTrees=10)
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",labels=labelIndexer.labels)
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, rf, labelConverter])
model = pipeline.fit(trainingData)
predictions = model.transform(testData)
evaluator = MulticlassClassificationEvaluator(labelCol="indexedCategory", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))
```

```
(14) Spark Jobs
output:  pyspark.sql.dataframe.DataFrame = [Address: string, Date: timestamp ... 26 more fields]
trainingData:  pyspark.sql.dataframe.DataFrame = [Address: string, Date: timestamp ... 26 more fields]
testData:  pyspark.sql.dataframe.DataFrame = [Address: string, Date: timestamp ... 26 more fields]
predictions:  pyspark.sql.dataframe.DataFrame = [Address: string, Date: timestamp ... 32 more fields]
Test Error = 0.646915
```

Results

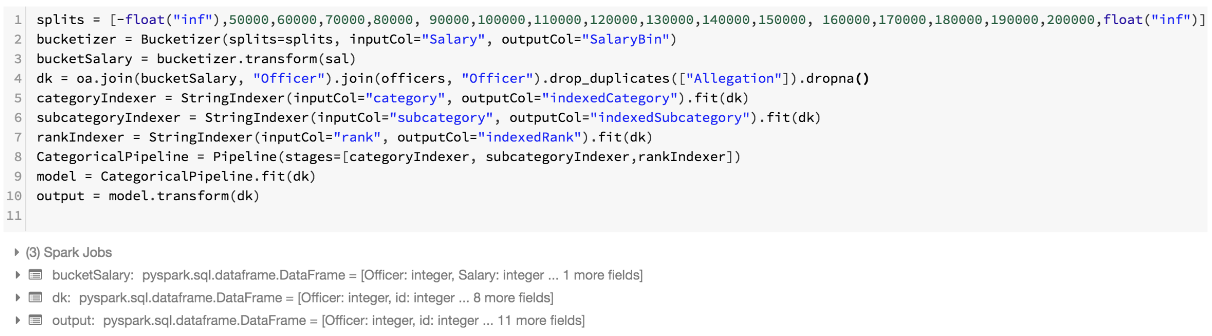
After a few runs of the same model, the average accuracy for predicting the allegation category was 65%. Given more time to incorporate more features of crimes such as subcategory, there could have been better results. ***We think this accuracy stems from leveraging a binary representation of the surrounding crimes on a given day and location of a complaint. Since there are over 20 crime types, that meant there were a large variety of possible one-hot encoding values. This combined with a large training set must have been positively reflected in the accuracy.***

Officer Salary

The second prediction we implemented was to predict the salary of an officer of a corresponding allegation. There is speculation that police officers are underpaid for the risks they take on a daily basis, so it would be interesting to see if there was a relationship between the type of allegation and the salary.

Code

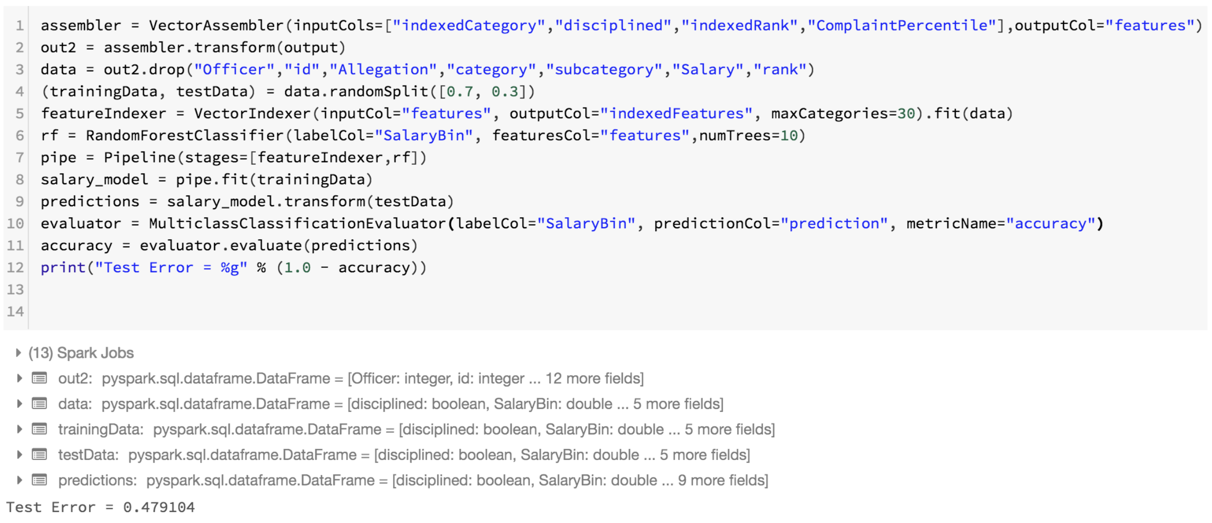
To prepare the data, we used a Spark Bucketizer to bucket the salaries of officers. This made the problem into a classification problem with a limited number of classes. Then, after enumerating the classification features for input, we put the features through a Random Forest Classifier.



```
splits = [-float("inf"),50000,60000,70000,80000, 90000,100000,110000,120000,130000,140000,150000, 160000,170000,180000,190000,200000,float("inf")]
bucketizer = Bucketizer(splits=splits, inputCol="Salary", outputCol="SalaryBin")
bucketSalary = bucketizer.transform(sal)
dk = oa.join(bucketSalary, "Officer").join(officers, "Officer").drop_duplicates(["Allegation"]).dropna()
categoryIndexer = StringIndexer(inputCol="category", outputCol="indexedCategory").fit(dk)
subcategoryIndexer = StringIndexer(inputCol="subcategory", outputCol="indexedSubcategory").fit(dk)
rankIndexer = StringIndexer(inputCol="rank", outputCol="indexedRank").fit(dk)
CategoricalPipeline = Pipeline(stages=[categoryIndexer, subcategoryIndexer,rankIndexer])
model = CategoricalPipeline.fit(dk)
output = model.transform(dk)

```

```
(3) Spark Jobs
bucketSalary:  pyspark.sql.dataframe.DataFrame = [Officer: integer, Salary: integer ... 1 more fields]
dk:  pyspark.sql.dataframe.DataFrame = [Officer: integer, id: integer ... 8 more fields]
output:  pyspark.sql.dataframe.DataFrame = [Officer: integer, id: integer ... 11 more fields]
```



```
assembler = VectorAssembler(inputCols=["indexedCategory","disciplined","indexedRank","ComplaintPercentile"],outputCol="features")
out2 = assembler.transform(output)
data = out2.drop("Officer","id","Allegation","category","subcategory","Salary","rank")
(trainingData, testData) = data.randomSplit([0.7, 0.3])
featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=30).fit(data)
rf = RandomForestClassifier(labelCol="SalaryBin", featuresCol="features",numTrees=10)
pipe = Pipeline(stages=[featureIndexer,rf])
salary_model = pipe.fit(trainingData)
predictions = salary_model.transform(testData)
evaluator = MulticlassClassificationEvaluator(labelCol="SalaryBin", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))
```

```
(13) Spark Jobs
out2:  pyspark.sql.dataframe.DataFrame = [Officer: integer, id: integer ... 12 more fields]
data:  pyspark.sql.dataframe.DataFrame = [disciplined: boolean, SalaryBin: double ... 5 more fields]
trainingData:  pyspark.sql.dataframe.DataFrame = [disciplined: boolean, SalaryBin: double ... 5 more fields]
testData:  pyspark.sql.dataframe.DataFrame = [disciplined: boolean, SalaryBin: double ... 5 more fields]
predictions:  pyspark.sql.dataframe.DataFrame = [disciplined: boolean, SalaryBin: double ... 9 more fields]
Test Error = 0.479104
```

Results

Again, after running the model a few times, the average accuracy for predicting the salary of an officer based on allegation data was less than 50%. When we ran the Bucketizer with splits every \$2500 rather than \$10000, the accuracy of the model actually increased by over 10%. Ultimately the best results seen were averaging 58%. ***As for intuitions on the results, we believe that this accuracy is due to the imbalance of data and the bucketing. Since at least 85% of officers were ranked as police officers with similar salaries, it may have been better to narrow the scope of the model and bucket solely police officers rather than every rank of officer. This would reduce our sample size by 15% or so, but this combined with a more granular bucket split, for example into bottom 25%, top 25% etc, the accuracy would increase. This most likely would have more potential in providing better insight for judging if having a lower salary for you rank reflected on higher complaints.***

Results and Analysis

Overall this experience was really beneficial. If we were to the checkpoint over again, we would expand our scope of features for both, potentially replacing the features used as input for the salary prediction. Also, we would have tested the data on a plethora of algorithms as we may have gotten better results with a different one. Of the two predictions, we felt the first one was relatively promising, and could be pursued further for a more competitive accuracy.

While we are not entirely satisfied with our results, this checkpoint provided us with working knowledge of Spark's capabilities for Machine Learning.