Sarah Ahmad (saz848)
Vikram Kohli (vkp260)
Junhao Li (jlo708)

Checkpoint 4: Machine Learning

Question 1. Is it possible to predict who complaints will come in against using a combination of officer information such as age, race, rank and demographics information of the complainants from past reports?

To answer this question, we attempted to create a machine learning application which predicted whether an officer had a complaint levied against them during or after 2009 as a proxy to simulate predicting whether future complaints would come in against an officer. The year 2009 was chosen in order to provide a sufficient portion of allegations for simulating the future allegations. Only active officers from the CPDB were used in the training and test sets and age was set to the officer's age in 2009. Much of the preprocessing was done in Trifacta; we used this to add columns into the database containing information including whether an officer had an allegation after 2009, as well as to calculate their age and to join the officer data table with the allegation data.

We ultimately decided on using four parameters in our machine learning algorithm, all of which relate to the social location of the officer; in particular, we chose the race of the officer, the gender, their position as a police officer and their age. We also filtered out officers with recent complaints to have a greater proportion of officers without recent complaints. We were initially encountering an issue where the percentage of officers in our dataset who had recently received a complaint was far greater than the number of officers who had not, which was causing our training data to be overwhelmingly skewed and therefore our decision tree was labeling all officers as likely to be involved in a complaint; we resolved this by removing a significant percentage of officers who were labeled as having received a recent complaint from the dataset. We split this data into training and test sets, and implemented a decision tree to attempt to predict whether an officer would be subject to another complaint based on this information.

Our initial decision tree algorithm reached a test accuracy of around 79%; it was able to predict whether or not an officer had been subject to a complaint after 2009 with a relatively high accuracy. Interestingly, we found that the first attribute that our decision tree split on was not race or gender, but rather age; an aspect of officer identities that we hadn't at all considered during our previous checkpoints. It appeared that officers particularly between the age ranges of 37.5 and 47.5 were predicted to be subject to complaints.

In order to connect our predictor to our previous research which centered largely around race and gender, we then decided to remove the ages of the officers from our dataset, and rerun the decision tree. This resulted in a slightly lower test accuracy of around 76%; which seems to suggest that, even without age, race gender and officer position still seem to provide strong predictive power for whether an officer is likely to be subject to a complaint. Here we found that the decision tree now initially split on gender as its first attribute, followed by race. In the case of both this decision tree and the other, it appeared that the status/position of the officer did not provide much information with regards to our question, and was not included as an attribute that was split over.

While we cannot be certain that our decision tree has captured an existing phenomenon, we believe that is the case. We are able to see the attributes splits within the tree and many of the tree outcomes do appear to be sensical representation of real trends. For example, it could be true that asian male officers below the age of 30 are very likely to receive allegations. However, the majority of officer are labeled as having recent complaints, making it easy for a decision tree to learn to mark most officers as due for a complaint.

Overall, it appears that the social location of officers can be used as an effective predictor of whether or not they are likely to be subject to a complaint. One interesting application of this could be to apply such a predictor to the dataset of all officers (who have yet to be subject to a complaint) and to see if over, for example, the next 5 years, the predictions made by the decision tree turn out to correctly identify officers that are ‘set for’ a complaint.

Question 2. Is it possible to predict whether an officer will be the subject of a settlement using the details of their complaint history including frequency of complaints and nature of complaints?

To create a model to make this prediction, we pulled the list of officers that had allegations in the CPDB and counted the number of allegations found for each officer. We then used the first complaint type for each officer to use as the value for nature of complaint for the officer. After that, we needed to get our decision value for our predictor which was whether the officer was involved in a settlement or not. We used Trifacta to check to see whether each officer was in the settlement database by joining on officer id. If the officer was, the value in our table would be one, and zero otherwise.

Initially, for our question we also wanted to use complainant demographics as a feature in our model. We decided on this because as we were trying to get our data ready for the machine learner, we found that we were unsure as to how to best represent the demographics so we decided that the frequency of complaints and nature of complaints would be sufficient as initial features. After exporting this table from Trifacta, we used Databricks to implement the machine learning algorithm. The data was split into training and testing data using 70% training and 30% testing data. We then applied a decision tree algorithm on this data, as our final output was binary, either the officer will be involved in a settlement or they won't.

Our initial decision tree accuracy was 92.7%. This looked great, but on closer inspection it turned out that our decision tree was only outputting a prediction of 0 for each split. We realized that this was because approximately 93% of the officers were not involved in a settlement and only 7% were involved. It seems like this may have been skewing our decision tree, and there's a chance that very little involved in settlement officers were being included in the testing set. Therefore, we chose to go back to Trifacta and clean the data further, but removing officers who were not involved in a settlement to get the ratio to be closer to 70% and 30% (no settlement, settlement).

After doing so, though our accuracy went down to 72.4%, we were getting predictions where officers would be involved in settlements. These cases were the following:

1. If the number of allegations was less than 4.5 and the officer was involved in bribery/official corruption
2. If the number of allegations was greater than 4.5 but less than 10.5
3. If the number of allegations was greater than 24.5, all types of complaints would result in a settlement except use of force, conduct unbecoming, and first amendment.

From looking at the decision tree that was created, it seems like the tree relied heavily on the number of allegations as opposed to the type of complaint. Though the tree did split on the type of complaint, the number of options of types was a lot greater for each split, and then the tree would go on to split by count again.

We again were able to see the output of our machine learner, which means that there is less of a feeling of a black-box with this approach. Also, it seems reasonable that a higher number of allegations would almost always yield a settlement, whereas a low count with a seemingly more serious offense would end up with a settlement.

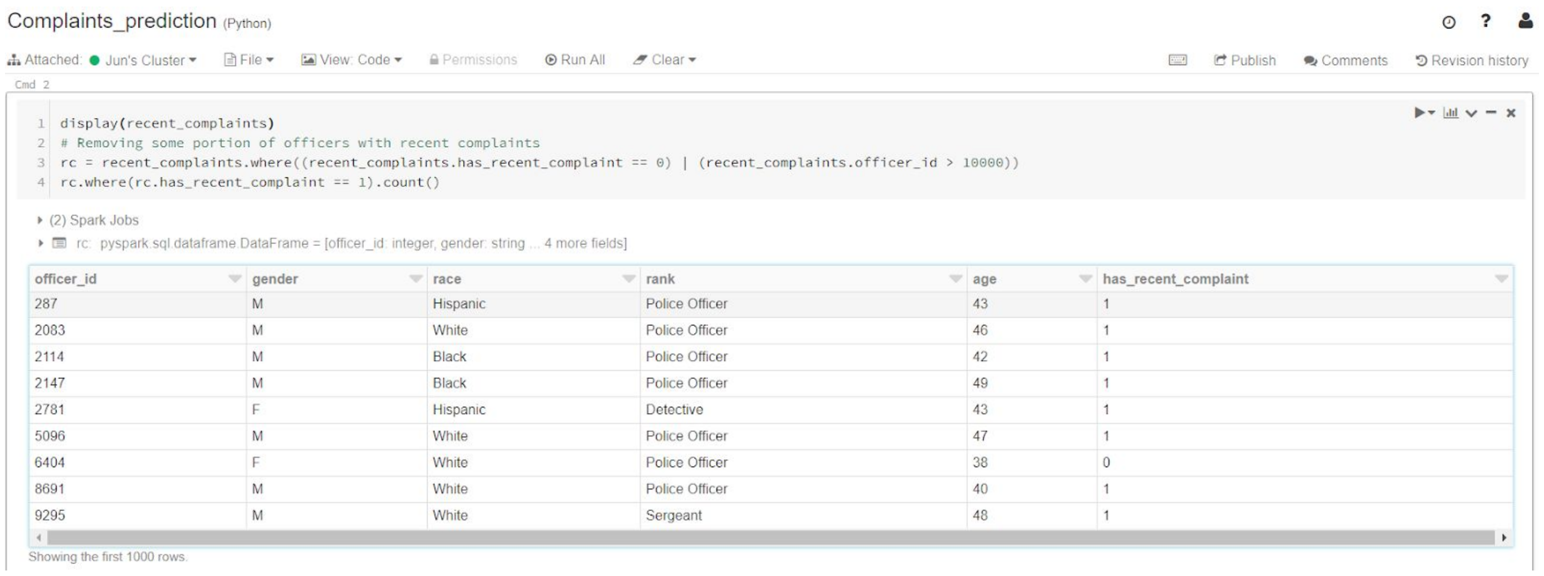
Previously in checkpoint two, we found that there wasn't a correlation between the amount that an officer owed in a settlement and the number of complaints an officer was involved in. However, our decision tree seems to rely heavily on the number of allegations to determine whether an officer will be involved in a settlement or not. From this, it seems that how much an officer owes in a settlement and whether an officer will be involved in a settlement are not related by number of allegations.

Results

Question 1:

Notebook link:
https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/4988437523563813/1129723808572205/3062227156193771/latest.html

Screenshots from Notebook:

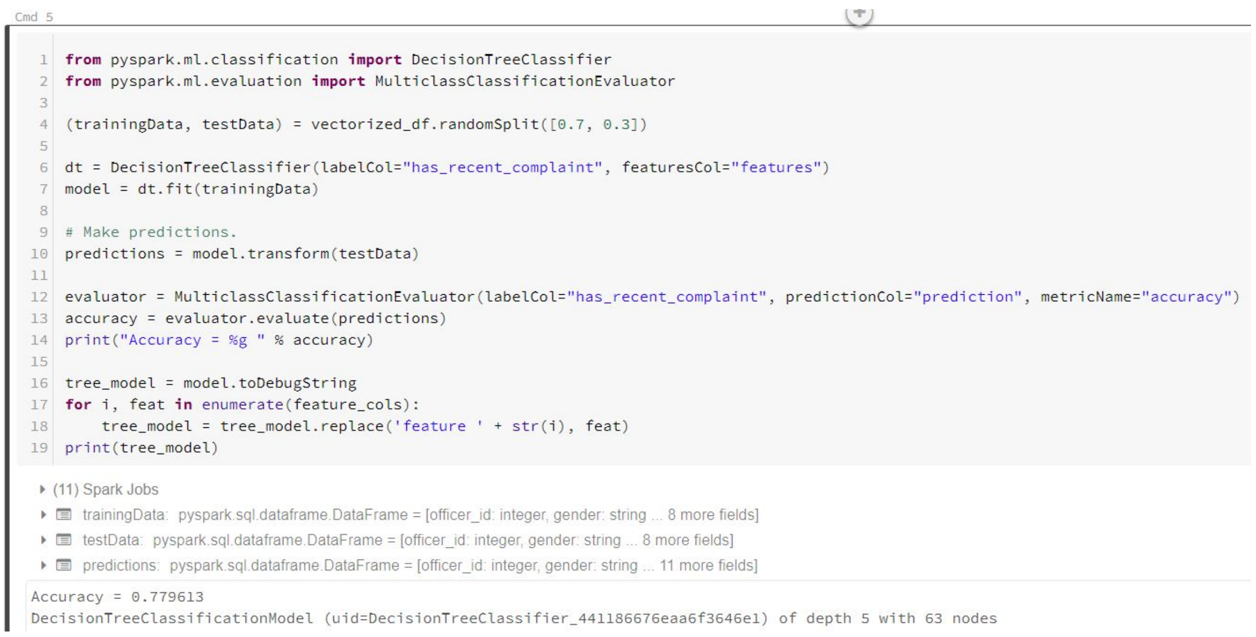


```
display(recent_complaints)
# Removing some portion of officers with recent complaints
rc = recent_complaints.where((recent_complaints.has_recent_complaint == 0) | (recent_complaints.officer_id > 10000))
rc.where(rc.has_recent_complaint == 1).count()
```

▸ (2) Spark Jobs
▸ rc: pyspark.sql.dataframe.DataFrame = [officer_id: integer, gender: string ... 4 more fields]

officer_id	gender	race	rank	age	has_recent_complaint
287	M	Hispanic	Police Officer	43	1
2083	M	White	Police Officer	46	1
2114	M	Black	Police Officer	42	1
2147	M	Black	Police Officer	49	1
2781	F	Hispanic	Detective	43	1
5096	M	White	Police Officer	47	1
6404	F	White	Police Officer	38	0
8691	M	White	Police Officer	40	1
9295	M	White	Sergeant	48	1

Showing the first 1000 rows.



```
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

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

dt = DecisionTreeClassifier(labelCol="has_recent_complaint", featuresCol="features")
model = dt.fit(trainingData)

# Make predictions.
predictions = model.transform(testData)

evaluator = MulticlassClassificationEvaluator(labelCol="has_recent_complaint", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Accuracy = %g " % accuracy)

tree_model = model.toDebugString
for i, feat in enumerate(feature_cols):
    tree_model = tree_model.replace('feature ' + str(i), feat)
print(tree_model)
```

▸ (11) Spark Jobs
▸ trainingData: pyspark.sql.dataframe.DataFrame = [officer_id: integer, gender: string ... 8 more fields]
▸ testData: pyspark.sql.dataframe.DataFrame = [officer_id: integer, gender: string ... 8 more fields]
▸ predictions: pyspark.sql.dataframe.DataFrame = [officer_id: integer, gender: string ... 11 more fields]

```
Accuracy = 0.779613
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_441186676eaa6f3646e1) of depth 5 with 63 nodes
```

Final Decision tree:

```
DecisionTreeClassificationModel
(uid=DecisionTreeClassifier_47b0b7da47cd39fb52d3) of depth 5 with 39 nodes
  If (gender_index in {1.0})
   If (rank_index in {8.0})
    Predict: 0.0
   Else (rank_index not in {8.0})
    If (race_index in {2.0,3.0,4.0})
     If (rank_index in {4.0,6.0,7.0})
      Predict: 1.0
     Else (rank_index not in {4.0,6.0,7.0})
      If (rank_index in {1.0,2.0})
       Predict: 1.0
      Else (rank_index not in {1.0,2.0})
       Predict: 1.0
    Else (race_index not in {2.0,3.0,4.0})
     If (rank_index in {4.0,5.0,6.0,7.0})
      If (rank_index in {7.0})
       Predict: 0.0
      Else (rank_index not in {7.0})
       Predict: 1.0
     Else (rank_index not in {4.0,5.0,6.0,7.0})
      If (rank_index in {2.0,3.0})
       Predict: 1.0
      Else (rank_index not in {2.0,3.0})
       Predict: 1.0
  Else (gender_index not in {1.0})
   If (rank_index in {2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,11.0})
    If (rank_index in {3.0,4.0,5.0,6.0,7.0,8.0,9.0,11.0})
     If (rank_index in {5.0,7.0,8.0,9.0,11.0})
      If (rank_index in {9.0,11.0})
       Predict: 0.0
      Else (rank_index not in {9.0,11.0})
       Predict: 1.0
     Else (rank_index not in {5.0,7.0,8.0,9.0,11.0})
      If (race_index in {1.0,2.0,3.0})
       Predict: 1.0
      Else (race_index not in {1.0,2.0,3.0})
       Predict: 1.0
    Else (rank_index not in {3.0,4.0,5.0,6.0,7.0,8.0,9.0,11.0})
     If (race_index in {4.0})
      Predict: 1.0
     Else (race_index not in {4.0})
      If (race_index in {1.0,2.0,3.0})
       Predict: 1.0
      Else (race_index not in {1.0,2.0,3.0})
       Predict: 1.0
```

```
Else (rank_index not in {2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,11.0})
 If (race_index in {1.0,2.0,3.0,4.0,5.0})
  If (race_index in {4.0,5.0})
   Predict: 1.0
  Else (race_index not in {4.0,5.0})
   If (race_index in {1.0,3.0})
    Predict: 1.0
   Else (race_index not in {1.0,3.0})
    Predict: 1.0
 Else (race_index not in {1.0,2.0,3.0,4.0,5.0})
  If (rank_index in {1.0})
   Predict: 1.0
  Else (rank_index not in {1.0})
   Predict: 1.0
```

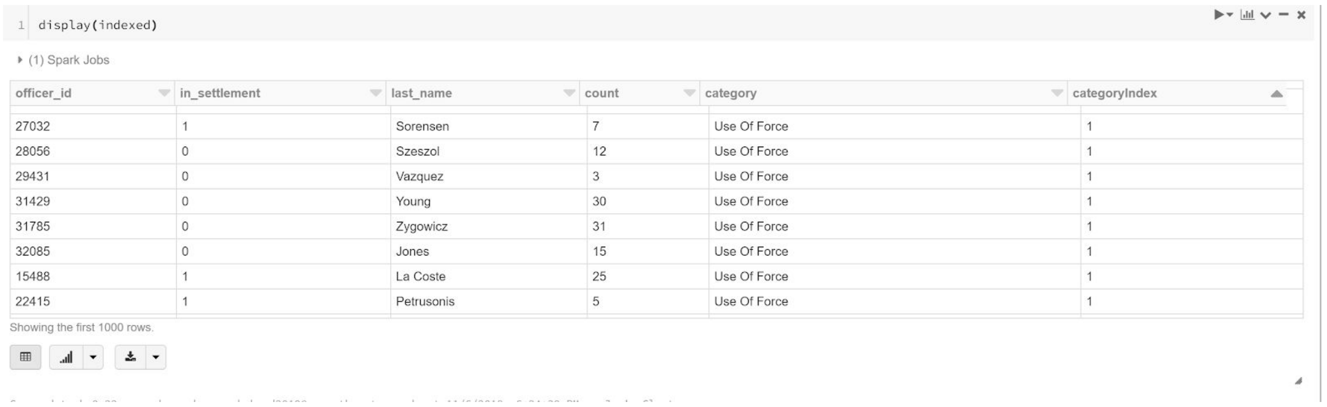
Question 2:

Notebook link:

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/4988437523563813/4338116524296920/4152000852923769/latest.html

Results:

Screenshots from notebook

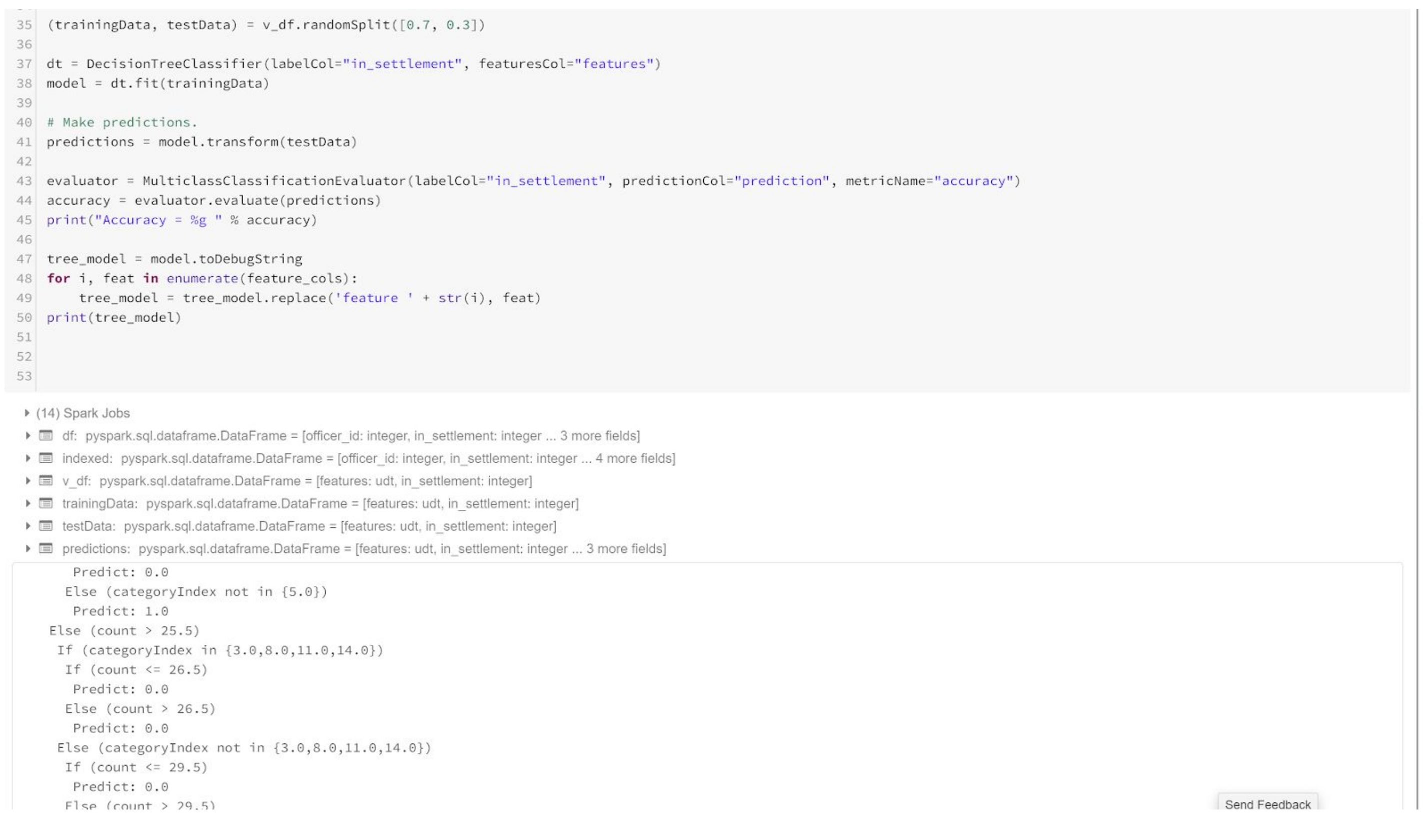


```
display(indexed)
```

(1) Spark Jobs

officer_id	in_settlement	last_name	count	category	categoryIndex
27032	1	Sorensen	7	Use Of Force	1
28056	0	Szeszol	12	Use Of Force	1
29431	0	Vazquez	3	Use Of Force	1
31429	0	Young	30	Use Of Force	1
31785	0	Zygowicz	31	Use Of Force	1
32085	0	Jones	15	Use Of Force	1
15488	1	La Coste	25	Use Of Force	1
22415	1	Petrusonis	5	Use Of Force	1

Showing the first 1000 rows.



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

dt = DecisionTreeClassifier(labelCol="in_settlement", featuresCol="features")
model = dt.fit(trainingData)

# Make predictions.
predictions = model.transform(testData)

evaluator = MulticlassClassificationEvaluator(labelCol="in_settlement", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Accuracy = %g " % accuracy)

tree_model = model.toDebugString
for i, feat in enumerate(feature_cols):
    tree_model = tree_model.replace('feature ' + str(i), feat)
print(tree_model)
```

(14) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [officer_id: integer, in_settlement: integer ... 3 more fields]

indexed: pyspark.sql.dataframe.DataFrame = [officer_id: integer, in_settlement: integer ... 4 more fields]

v_df: pyspark.sql.dataframe.DataFrame = [features: udt, in_settlement: integer]

trainingData: pyspark.sql.dataframe.DataFrame = [features: udt, in_settlement: integer]

testData: pyspark.sql.dataframe.DataFrame = [features: udt, in_settlement: integer]

predictions: pyspark.sql.dataframe.DataFrame = [features: udt, in_settlement: integer ... 3 more fields]

```
       Predict: 0.0
      Else (categoryIndex not in {5.0})
       Predict: 1.0
    Else (count > 25.5)
     If (categoryIndex in {3.0,8.0,11.0,14.0})
      If (count <= 26.5)
       Predict: 0.0
      Else (count > 26.5)
       Predict: 0.0
     Else (categoryIndex not in {3.0,8.0,11.0,14.0})
      If (count <= 29.5)
       Predict: 0.0
      Else (count > 29.5)
```

Final Decision Tree

DecisionTreeClassificationModel (uid=DecisionTreeClassifier_40ccb58e8d28d21b4c24) of depth 5 with 53 nodes

```
 If (count <= 19.5)
  If (categoryIndex in {12.0,13.0})
   If (count <= 10.5)
    If (count <= 4.5)
     If (categoryIndex in {13.0})
      Predict: 0.0
     Else (categoryIndex not in {13.0})
      Predict: 1.0
    Else (count > 4.5)
     Predict: 1.0
   Else (count > 10.5)
    Predict: 0.0
  Else (categoryIndex not in {12.0,13.0})
   If (categoryIndex in {9.0})
    If (count <= 16.5)
     If (count <= 1.5)
      Predict: 0.0
     Else (count > 1.5)
      Predict: 0.0
    Else (count > 16.5)
     Predict: 0.0
   Else (categoryIndex not in {9.0})
    If (count <= 2.5)
     If (categoryIndex in {5.0,6.0,10.0,11.0})
      Predict: 0.0
     Else (categoryIndex not in {5.0,6.0,10.0,11.0})
      Predict: 0.0
    Else (count > 2.5)
     If (categoryIndex in {6.0,8.0,10.0,11.0})
      Predict: 0.0
     Else (categoryIndex not in {6.0,8.0,10.0,11.0})
      Predict: 0.0
 Else (count > 19.5)
```

```
   If (categoryIndex in {2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11.0,12.0,14.0})
    If (count <= 25.5)
     If (count <= 24.5)
      If (categoryIndex in {3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11.0,12.0,14.0})
       Predict: 0.0
      Else (categoryIndex not in
{3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11.0,12.0,14.0})
       Predict: 0.0
     Else (count > 24.5)
      If (categoryIndex in {5.0})
       Predict: 0.0
      Else (categoryIndex not in {5.0})
       Predict: 1.0
    Else (count > 25.5)
     If (categoryIndex in {3.0,8.0,11.0,14.0})
      If (count <= 26.5)
       Predict: 0.0
      Else (count > 26.5)
       Predict: 0.0
     Else (categoryIndex not in {3.0,8.0,11.0,14.0})
      If (count <= 29.5)
       Predict: 0.0
      Else (count > 29.5)
       Predict: 0.0
   Else (categoryIndex not in
{2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11.0,12.0,14.0})
    If (count <= 24.5)
     If (categoryIndex in {0.0})
      If (count <= 20.5)
       Predict: 0.0
      Else (count > 20.5)
       Predict: 0.0
     Else (categoryIndex not in {0.0})
      If (count <= 22.5)
       Predict: 0.0
      Else (count > 22.5)
       Predict: 0.0
```

```
Else (count > 24.5)
 If (count <= 26.5)
  If (categoryIndex in {1.0})
   Predict: 0.0
  Else (categoryIndex not in {1.0})
   Predict: 0.0
 Else (count > 26.5)
  If (count <= 29.5)
   Predict: 0.0
  Else (count > 29.5)
   Predict: 0.0
```