databricksAssignment

Logistic Regression Classifier

Module 4 Assignment

This final assignment is broken up into 2 parts:

1. Completing this Logistic Regression Classifier notebook
   - Submitting question answers to Coursera
   - Uploading notebook to Coursera for peer reviewing
2. Answering 3 free response questions on Coursera platform

In this notebook you:

- Preprocess data for use in a machine learning model
- Step through creating a sklearn logistic regression model for classification
- Predict the `Call_Type_Group` for incidents in a SQL table

For each **bold** question, input its answer in Coursera.

```
%run ../Includes/Classroom-Setup
```

Data mounted to /mnt/davis ...

OK

Load the `/mnt/davis/fire-calls/fire-calls-clean.parquet` data as `fireCallsClean` table.

```sql
-- TODO
USE DATABRICKS;
CREATE TABLE IF NOT EXISTS fireCallsClean
USING Parquet
OPTIONS (path "/mnt/davis/fire-calls/fire-calls-clean.parquet")
```

OK

Check that your data is loaded in properly.

```sql
SELECT * FROM fireCallsClean LIMIT 10
```

	Call_Number	Unit_ID	Incident_Number	Call_Type	Call_Date
1	141600888	65	14055109	Traffic Collision	06/09/2014
2	162743687	E01	16108733	Medical Incident	09/30/2016
3	102210202	75	10069623	Medical Incident	08/09/2010
4	160681260	E42	16027085	Medical Incident	03/08/2016
5	113200298	E18	11106370	Structure Fire	11/16/2011
6	162584030	KM07	16101787	Medical Incident	09/14/2016
7	133150239	KM09	13107112	Medical Incident	11/11/2013
8	170553524	66	17023871	Medical Incident	02/24/2017

Showing all 10 rows.

By the end of this assignment, we would like to train a logistic regression model to predict 2 of the most common `Call_Type_Group` given information from the rest of the table.

Write a query to see what the different `Call_Type_Group` values are and their respective counts.

Question 1

How many calls of `Call_Type_Group` "Fire"?

```sql
SELECT Call_Type_Group, COUNT(*) FROM fireCallsClean
GROUP BY `Call_Type_Group`
--4196
```

	Call_Type_Group	count(1)
1	Alarm	32566
2	null	246459
3	Potentially Life-Threatening	78030
4	Non Life-threatening	56168
5	Fire	4196

Showing all 5 rows.

```
SELECT COUNT(*) FROM fireCallsClean AS
(SELECT * FROM )
```

	count(1)
1	417419

Showing all 1 rows.

Let's drop all the rows where `Call_Type_Group = null`. Since we don't have a lot of `Call_Type_Group` with the value `Alarm` and `Fire`, we will also drop these calls from the table. Call this new temporary view `fireCallsGroupCleaned`.

```
DROP TABLE IF EXISTS fireCallsGroupCleaned;
CREATE TABLE fireCallsGroupCleaned AS
  (SELECT * FROM fireCallsClean WHERE Call_Type_Group <> '' AND
  Call_Type_Group <> 'Alarm' AND
  Call_Type_Group <> 'Fire')
```

OK

```
SELECT * FROM fireCallsGroupCleaned
```

	Call_Number	Unit_ID	Incident_Number	Call_Type	Call_Date
1	142443718	89	14085209	Medical Incident	09/01/2014
2	163481705	64	16139301	Medical Incident	12/13/2016
3	150993416	85	15037632	Medical Incident	04/09/2015
4	183031493	E23	18127022	Medical Incident	10/30/2018
5	171863111	E25	17078500	Medical Incident	07/05/2017

7	150221689	52	15008569	Medical Incident	01/22/2015
8	121170333	E19	12038908	Medical Incident	04/26/2012

Showing the first 1000 rows.

Check that every entry in `fireCallsGroupCleaned` has a `Call_Type_Group` of either `Potentially Life-Threatening` or `Non Life-threatening`.

```
SELECT COUNT(*) FROM FireCallsGroupCleaned
WHERE Call_Type_Group != 'Potentially Life-Threatening' AND
Call_Type_Group != 'Not Life-threatening'
--No, some rows contains other values
```

	count(1)
1	56168

Showing all 1 rows.

Question 2

How many rows are in `fireCallsGroupCleaned`?

```
SELECT COUNT(*) FROM fireCallsGroupCleaned
```

	count(1)
1	134198

Showing all 1 rows.

We probably don't need all the columns of `fireCallsGroupCleaned` to make our prediction. Select the following columns from `fireCallsGroupCleaned` and create a view called `fireCallsDF` so we can access this table in Python:

- "Call_Type"

- "Fire_Prevention_District"
- "Neighborhooods_-_Analysis_Boundaries"
- "Number_of_Alarms"
- "Original_Priority"
- "Unit_Type"
- "Battalion"
- "Call_Type_Group"

```
CREATE TABLE fireCallsDF AS
(SELECT Call_Type, Fire_Prevention_District, `Neighborhooods_-
_Analysis_Boundaries`, Number_of_Alarms, Original_Priority,
  Unit_Type, Battalion, Call_Type_Group
FROM FireCallsGroupCleaned)
```

OK

```
SELECT * FROM fireCallsDF
```

	Call_Type	Fire_Prevention_District	Neighborhooods_-_Analysis_Boundarie
1	Traffic Collision	10	Bayview Hunters Point
2	Medical Incident	2	South of Market
3	Medical Incident	10	Portola
4	Medical Incident	3	South of Market
5	Medical Incident	2	Mission
6	Medical Incident	8	Sunset/Parkside
7	Medical Incident	6	Castro/Upper Market
8	Medical Incident	4	Nob Hill

Showing the first 1000 rows.

Fill in the string SQL statement to load the `fireCallsDF` table you just created into python.

```
%python
# TODO
df = sql("SELECT * FROM fireCallsDF")
display(df)
```

	Call_Type	Fire_Prevention_District	Neighborhooods_-_Analysis_Boundarie
1	Traffic Collision	10	Bayview Hunters Point

2	Medical Incident	2	South of Market
3	Medical Incident	10	Portola
4	Medical Incident	3	South of Market
5	Medical Incident	2	Mission
6	Medical Incident	8	Sunset/Parkside
7	Medical Incident	6	Castro/Upper Market
8	Medical Incident	4	Nob Hill

Showing the first 1000 rows.

Creating a Logistic Regression Model in Sklearn

First we will convert the Spark DataFrame to pandas so we can use sklearn to preprocess the data into numbers so that it is compatible with the logistic regression algorithm with a LabelEncoder (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html).

Then we'll perform a train test split on our pandas DataFrame. Remember that the column we are trying to predict is the `Call_Type_Group`.

```
%python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

pdDF = df.toPandas()
le = LabelEncoder()
numerical_pdDF = pdDF.apply(le.fit_transform)

X = numerical_pdDF.drop("Call_Type_Group", axis=1)
y = numerical_pdDF["Call_Type_Group"].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Look at our training data `X_train` which should only have numerical values now.

```
%python
display(X_train)
```

	Call_Type	Fire_Prevention_District	Neighborhooods_-_Analysis_Boundaries
1	0	4	27
2	0	8	35
3	0	3	34
4	0	4	27
5	0	0	3
6	0	1	0
7	0	2	36
8	0	2	9

Showing the first 1000 rows.

We'll create a pipeline with 2 steps.

1. One Hot Encoding (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html Converts our features into vectorized features by creating a dummy column for each value in that category.

2. Logistic Regression model (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.htr Although the name includes "regression", it is used for classification by predicting the probability that the `Call Type Group` is one label and not the other.

```
%python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline

ohe = ("ohe", OneHotEncoder(handle_unknown="ignore"))
lr = ("lr", LogisticRegression())

pipeline = Pipeline(steps = [ohe, lr]).fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
```

```
/databricks/python/lib/python3.6/site-packages/sklearn/linear_model/logisti
c.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22.
Specify a solver to silence this warning.
  FutureWarning)
```

Run the following cell to see how well our model performed on test data (data that wasn't used to train the model)!

```
%python
from sklearn.metrics import accuracy_score
print(f"Accuracy of model: {accuracy_score(y_pred, y_test)}")
```

```
Accuracy of model: 0.8163934426229508
```

Question 3

What is the accuracy of our model on test data? Round to the nearest percent.

Save pipeline (with both stages) to disk.

```
%python
import mlflow
from mlflow.sklearn import save_model

model_path = "/dbfs/" + username + "/Call_Type_Group_lr"
dbutils.fs.rm(username + "/Call_Type_Group_lr", recurse=True)
save_model(pipeline, model_path)
```

UDF

Now that we have created and trained a machine learning pipeline, we will use MLflow to register the `.predict` function of the sklearn pipeline as a UDF which we can use later to apply in parallel. Now we can refer to this with the name `predictUDF` in SQL.

```
%python
import mlflow
from mlflow.pyfunc import spark_udf

predict = spark_udf(spark, model_path, result_type="int")
spark.udf.register("predictUDF", predict)
```

```
Out[8]: <function mlflow.pyfunc.spark_udf.<locals>.predict(*args)>
```

Create a view called `testTable` of our test data `X_test` so that we can see this table in SQL.

```
%python
spark_df = spark.createDataFrame(X_test)
spark_df.createOrReplaceTempView("testTable")
```

Create a table called `predictions` using the `predictUDF` function we registered beforehand. Apply the `predictUDF` to every row of `testTable` in parallel so that each row of `testTable` has a `Call_Type_Group` prediction.

```
SELECT * FROM testTable
```

	Call_Type	Fire_Prevention_District	Neighborhooods_-_Analysis_Boundaries
1	0	9	40
2	2	1	0
3	0	2	9
4	0	2	18
5	0	2	18
6	0	2	18
7	0	8	35
8	0	3	34

Showing the first 1000 rows.

```
-- TODO
USE DATABRICKS;
CREATE TABLE predictions AS (
  SELECT *, CAST(predictUDF(Call_Type,
Fire_Prevention_District,`Neighborhooods_-
_Analysis_Boundaries`,Number_of_Alarms,
  Original_Priority, Unit_Type, Battalion
 ) as double)as prediction
  FROM testTable)
```

OK

Now take a look at the table and see what your model predicted for each call entry!

```
SELECT * FROM predictions LIMIT 10
```

	Call_Type	Fire_Prevention_District	Neighborhooods_-_Analysis_Boundaries
1	0	2	18
2	0	3	34
3	0	5	2
4	0	2	18
5	0	4	27
6	0	4	12
7	0	3	36
8	0	6	21

Showing all 10 rows.

Question 4:

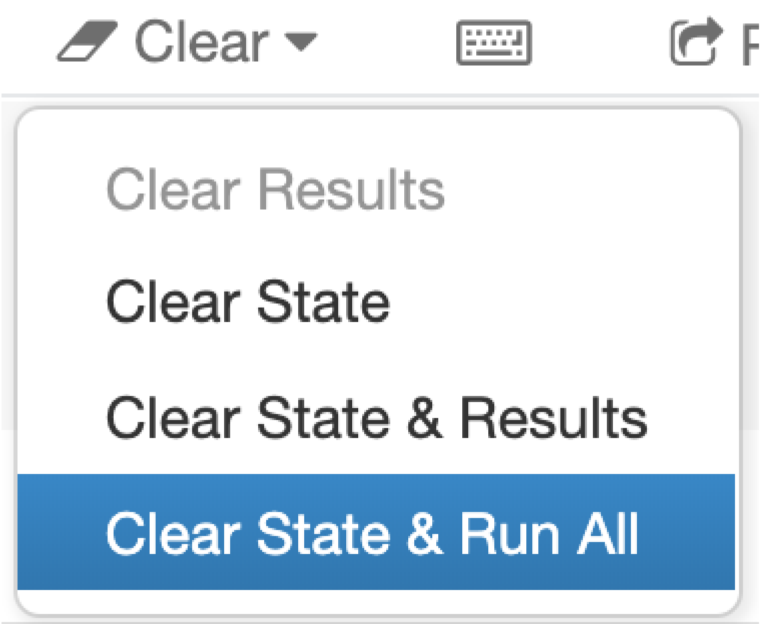
What 2 values are in the `prediction` column?

```
-- 1 and 0
```

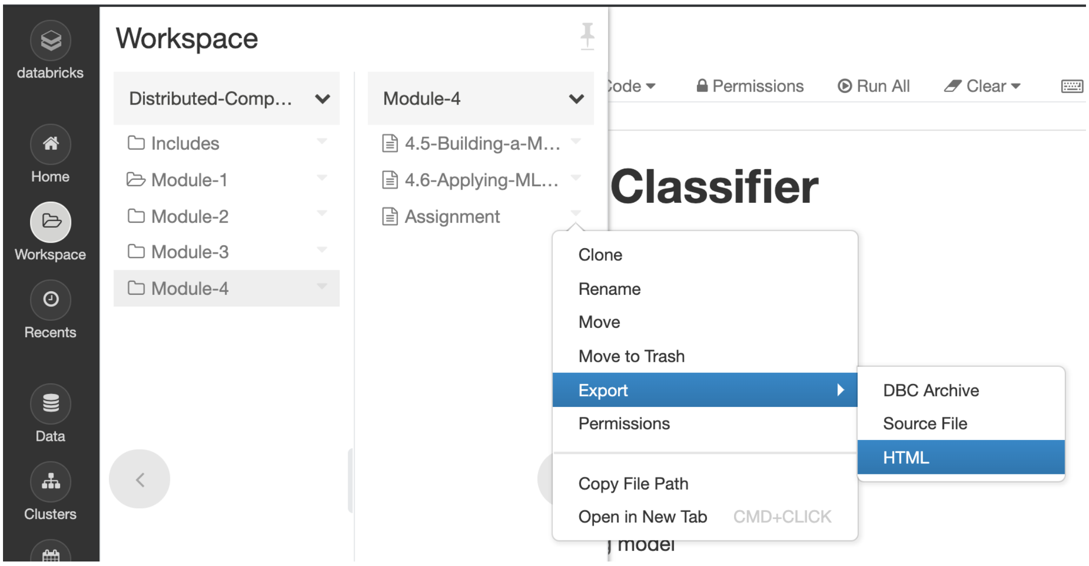
Congrats on finishing your last assignment notebook!

Now you will have to upload this notebook to Coursera for peer reviewing.

1. Make sure that all your code will run without errors
   - Check this by clicking the "Clear State & Run All" dropdown option at the top of your notebook



2. Click on the "Workspace" icon on the side bar
3. Next to the notebook you're working in right now, click on the dropdown arrow
4. In the dropdown, click on "Export" then "HTML"



5. On the Coursera platform, upload this HTML file to Week 4's Peer Review Assignment

Go back onto the Coursera platform for the free response portion of this assignment and for instructions on how to review your peer's work.