databricks Assignment

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.

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

OK

Check that your data is loaded in properly.

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

Call_Number	Unit_ID	Incident_Number	Call_Type	Call_Date	Watch_Date
141600888	65	14055109	Traffic Collision	06/09/2014	06/09/2014
162743687	E01	16108733	Medical Incident	09/30/2016	09/30/2016
102210202	75	10069623	Medical Incident	08/09/2010	08/09/2010
160681260	E42	16027085	Medical	03/08/2016	03/08/2016

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"?

```
-- TODO
SELECT `Call_Type_Group` AS Call_Type_Group,
    COUNT(`Call_Type_Group`) AS count
FROM fireCallsClean
GROUP BY Call_Type_Group
```

Call_Type_Group
Alarm
null
Potentially Life-Threatening
Non Life-threatening
Fire

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 table `fireCallsGroupCleaned`.

```
-- TODO
CREATE TABLE IF NOT EXISTS fireCallsGroupCleaned AS
(SELECT *
FROM fireCallsClean
WHERE Call_Type_Group IN('Potentially Life-Threatening', 'Non Life-
threatening')
)
```

OK

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

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

count(1)
134198

Question 2

How many rows are in `fireCallsGroupCleaned`?

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"

```
-- TODO
CREATE OR REPLACE VIEW 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

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_Boundaries
Medical Incident	3	Tenderloin
Medical Incident	8	Sunset/Parkside
Medical Incident	1	Tenderloin
Medical Incident	2	Hayes Valley
Medical Incident	4	Western Addition
Medical Incident	2	Mission
Medical Incident	2	South of Market

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_Boundarie
0	2	36
0	2	9
0	2	36
0	0	23
0	0	36

0	3	5
0	7	26

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.htr 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.h 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/logist
ic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. 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: 0.8179955290611028
```

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[17]: <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
X_test.to_csv("/dbfs/" + username + "/Call_Type_Group_lr"+ "/test.csv")
dbutils.fs.ls("/jiawen.bian@mail.mcgill.ca/Call_Type_Group_lr/")


Out[21]: [FileInfo(path='dbfs:/jiawen.bian@mail.mcgill.ca/Call_Type_Group_
lr/MLmodel', name='MLmodel', size=281),
 FileInfo(path='dbfs:/jiawen.bian@mail.mcgill.ca/Call_Type_Group_lr/conda.
yaml', name='conda.yaml', size=130),
 FileInfo(path='dbfs:/jiawen.bian@mail.mcgill.ca/Call_Type_Group_lr/model.
pkl', name='model.pkl', size=2636)]

CREATE OR REPLACE TEMPORARY VIEW testTable
USING csv
OPTIONS (
    header 'true',
    path "/jiawen.bian@mail.mcgill.ca/Call_Type_Group_lr/test.csv",
    inferSchema "true"
   )
```

OK

```
SELECT *
FROM testTable
```

_c0	Call_Type	Fire_Prevention_District	Neighborhooods_-_Analysis_Bo
122451	0	8	13
14029	2	0	23
51449	0	0	23

17981	0	8	40
66284	0	5	9
75991	0	2	2
15962	0	5	8

Showing the first 1000 rows.

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.

```
-- TODO
USE DATABRICKS;
DROP TABLE IF EXISTS predictions;

CREATE TEMPORARY VIEW predictions AS (
  SELECT cast(predictUDF(Call_Type, Fire_Prevention_District,
`Neighborhooods_-_Analysis_Boundaries`,
                Number_of_Alarms, Original_Priority, Unit_Type, Battalion)
as int) as Call_Type_Group, *
  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_Group	_c0	Call_Type	Fire_Prevention_District	Neighborhooods
0	122451	0	8	13
1	14029	2	0	23

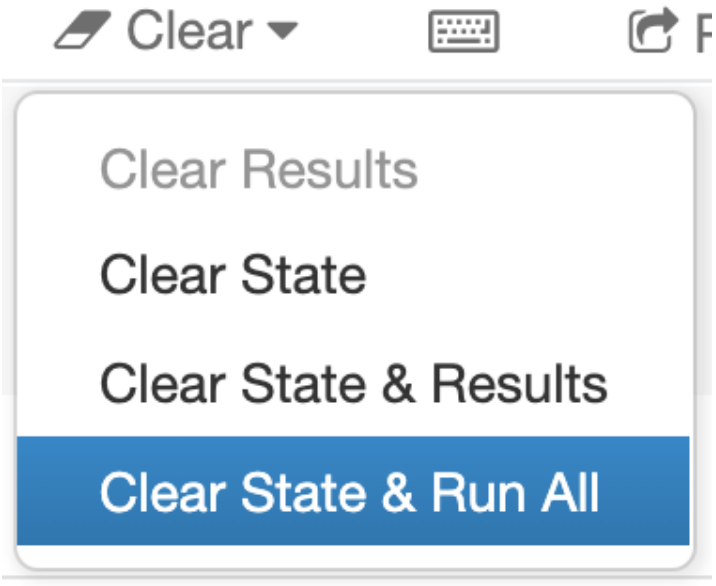
0	51449	0	0	23
1	17981	0	8	40
0	66284	0	5	9
0	75991	0	2	2
0	15962	0	5	8

Question 4:

What 2 values are in the `prediction` column?

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