ML_with_Spark

April 24, 2023

0.1 Set up bucket and start pyspark session

[1]:
```
PROJECT = !gcloud config get-value project
PROJECT = PROJECT[0]
BUCKET = PROJECT + "-dsongcp"
import os
os.environ["BUCKET"] = BUCKET
```

[2]:
```
from pyspark.sql import SparkSession
from pyspark import SparkContext

sc = SparkContext("local", "logistic")
spark = SparkSession .builder .appName("Logistic regression w/ Spark ML") .
 ↪getOrCreate()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
23/04/24 21:14:29 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
23/04/24 21:14:29 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
23/04/24 21:14:29 INFO org.apache.spark.SparkEnv: Registering
BlockManagerMasterHeartbeat
23/04/24 21:14:29 INFO org.apache.spark.SparkEnv: Registering
OutputCommitCoordinator
```

0.2 Create a Spark Dataframe for training

[3]:
```
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
from pyspark.mllib.regression import LabeledPoint
```

0.3 Read and clean up dataset

[5]:
```
traindays = spark.read .option("header", "true") .csv("gs://{}/flights/trainday.
 ↪csv".format(BUCKET))
traindays.createOrReplaceTempView("traindays")
```

[6]:
```
sql = """
SELECT * FROM traindays LIMIT 5"""
spark.sql(sql).show()
```

```
+----------+------------+
|   FL_DATE|is_train_day|
+----------+------------+
|2015-01-01|        True|
|2015-01-02|       False|
|2015-01-03|       False|
|2015-01-04|        True|
|2015-01-05|        True|
+----------+------------+
```

[7]:
```
inputs = 'gs://{}/flights/tzcorr/all_flights-00000-*'.format(BUCKET)
```

[8]:
```
flights = spark.read.json(inputs)
flights.createOrReplaceTempView('flights')
```

[9]:
```
trainquery = """
SELECT
  DEP_DELAY, TAXI_OUT, ARR_DELAY, DISTANCE
FROM flights f
JOIN traindays t
ON f.FL_DATE == t.FL_DATE
WHERE
  t.is_train_day == 'True'
"""
traindata = spark.sql(trainquery)
```

[13]:
```
print(traindata.head(3))
```

```
[Row(DEP_DELAY=-3.0, TAXI_OUT=14.0, ARR_DELAY=-16.0, DISTANCE='370.00'),
Row(DEP_DELAY=24.0, TAXI_OUT=12.0, ARR_DELAY=12.0, DISTANCE='370.00'),
Row(DEP_DELAY=3.0, TAXI_OUT=31.0, ARR_DELAY=10.0, DISTANCE='370.00')]
```

[14]:
```
traindata.describe().show()
```

```
[Stage 13:>                                                         (0 + 1) / 1]
+-------+------------------+------------------+------------------+--------------
---+
|summary|         DEP_DELAY|          TAXI_OUT|         ARR_DELAY|
DISTANCE|
+-------+------------------+------------------+------------------+--------------
```

```
---+
|  count|              46439|              46422|              46355|
46936|
|   mean|
8.561769202609876|15.427685149282668|3.2853413871211306|916.0707133117437|
| stddev|30.752752455053308| 8.427384168645757|
32.98848343691196|591.9164453757172|
|    min|              -22.0|                2.0|              -77.0|
1009.00|
|    max|              711.0|              178.0|              719.0|
980.00|
+-------+------------------+------------------+------------------+-------------
---+
```

0.4 Clean the dataset

[15]:
```
trainquery = """
SELECT
DEP_DELAY, TAXI_OUT, ARR_DELAY, DISTANCE
FROM flights f
JOIN traindays t
ON f.FL_DATE == t.FL_DATE
WHERE
t.is_train_day == 'True' AND
f.dep_delay IS NOT NULL AND
f.arr_delay IS NOT NULL
"""
traindata = spark.sql(trainquery)
traindata.describe().show()
```

```
[Stage 17:>                                                         (0 + 1) / 1]
+-------+------------------+------------------+------------------+-------------
---+
|summary|         DEP_DELAY|          TAXI_OUT|         ARR_DELAY|
DISTANCE|
+-------+------------------+------------------+------------------+-------------
---+
|  count|              46355|              46355|              46355|
46355|
|   mean| 8.539531873584295|15.421507927947363|3.2853413871211306|
917.660230827311|
| stddev|30.700034730525516|  8.41130660980497|
32.98848343691196|592.0960248192869|
|    min|              -22.0|                2.0|              -77.0|
1009.00|
```

```
|    max|             711.0|             178.0|             719.0|
980.00|
+-------+------------------+------------------+------------------+-------------
---+
```

The table shows that there are some issues with the data. Not all of the records have values for all of the variables, there are different count stats for DEP_DELAY, TAXI_OUT, ARR_DELAY and DISTANCE. This happens because:

Flights are scheduled but never depart

Some depart but are cancelled before take off

Some flights are diverted and therefore never arrive

[16]:
```
# Remove flights that have been cancelled or diverted using the following query
trainquery = """
SELECT
  DEP_DELAY, TAXI_OUT, ARR_DELAY, DISTANCE
FROM flights f
JOIN traindays t
ON f.FL_DATE == t.FL_DATE
WHERE
  t.is_train_day == 'True' AND
  f.CANCELLED == 'False' AND
  f.DIVERTED == 'False'
"""
traindata = spark.sql(trainquery)
traindata.describe().show()
```

```
[Stage 21:>                                                         (0 + 1) / 1]
+-------+------------------+------------------+------------------+-------------
---+
|summary|         DEP_DELAY|          TAXI_OUT|         ARR_DELAY|
DISTANCE|
+-------+------------------+------------------+------------------+-------------
---+
|  count|             46355|             46355|             46355|
46355|
|   mean| 8.539531873584295|15.421507927947363|3.2853413871211306|
917.660230827311|
| stddev|30.700034730525516|  8.41130660980497|
32.98848343691196|592.0960248192869|
|    min|             -22.0|               2.0|             -77.0|
1009.00|
|    max|             711.0|             178.0|             719.0|
980.00|
```

```
+-------+-----------------+-----------------+-----------------+-------------
---+
```

0.5 Develop a logistic regression model

[17]:
```
def to_example(fields):
    return LabeledPoint(\
               float(fields['ARR_DELAY'] < 15), #ontime? \
               [ \
                   fields['DEP_DELAY'], \
                   fields['TAXI_OUT'],  \
                   fields['DISTANCE'],  \
               ])
```

[18]:
```
# Map this training example function to the training dataset
examples = traindata.rdd.map(to_example)
```

[19]:
```
# provide a training DataFrame for the Spark logistic regression module:
lrmodel = LogisticRegressionWithLBFGS.train(examples, intercept=True)
```

```
23/04/24 21:24:06 WARN com.github.fommil.netlib.BLAS: Failed to load
implementation from: com.github.fommil.netlib.NativeSystemBLAS
23/04/24 21:24:06 WARN com.github.fommil.netlib.BLAS: Failed to load
implementation from: com.github.fommil.netlib.NativeRefBLAS
```

[20]:
```
# val
print(lrmodel.weights,lrmodel.intercept)
```

```
[-0.17926510230641074,-0.1353410840270897,0.00047781052266304745]
5.403405250989946
```

[22]:
```
# testing
# A departure delay of 6 minutes
# A taxi-out time of 12 minutes
# A flight distance of 594 miles

print(lrmodel.predict([6.0,12.0,594.0]))
```

```
1
```

[25]:
```
# testing
# A departure delay of 36 minutes
# A taxi-out time of 12 minutes
# A flight distance of 594 miles
```

```
print(lrmodel.predict([36.0,12.0,594.0]))
```

0

[26]:
```
# checking prediction probability values
lrmodel.clearThreshold()
print(lrmodel.predict([6.0,12.0,594.0]))
print(lrmodel.predict([36.0,12.0,594.0]))
```

0.9520080900763146
0.08390675828170738

[27]:
```
# Setting threshold as 0.7, if above 0.7, will consider as 1, otherwise 0
lrmodel.setThreshold(0.7)
print(lrmodel.predict([6.0,12.0,594.0]))
print(lrmodel.predict([36.0,12.0,594.0]))
```

1
0

0.6 Save and restore a logistic regression model

[28]:
```
MODEL_FILE='gs://' + BUCKET + '/flights/sparkmloutput/model'
os.system('gsutil -m rm -r ' + MODEL_FILE)
```

CommandException: 1 files/objects could not be removed.

[28]: 256

[29]:
```
lrmodel.save(sc, MODEL_FILE)
print('{} saved'.format(MODEL_FILE))
```

gs://qwiklabs-gcp-02-78ea052d93d3-dsongcp/flights/sparkmloutput/model saved

[30]:
```
lrmodel = 0
print(lrmodel)
```

0

[31]:
```
from pyspark.mllib.classification import LogisticRegressionModel
lrmodel = LogisticRegressionModel.load(sc, MODEL_FILE)
lrmodel.setThreshold(0.7)
```

0.7 Predict with the logistic regression model

[35]:
```
lrmodel.clearThreshold()
print(lrmodel.predict([36.0,12.0,594.0]))
print(lrmodel.predict([8.0,4.0,594.0]))

lrmodel.setThreshold(0.7)
print(lrmodel.predict([36.0,12.0,594.0]))
print(lrmodel.predict([8.0,4.0,594.0]))
```

```
0.08390675828170738
0.9761478464217875
0
1
```

0.8 Examine model behavior

[36]:
```
lrmodel.clearThreshold() # to make the model produce probabilities
print(lrmodel.predict([20, 10, 500]))
```

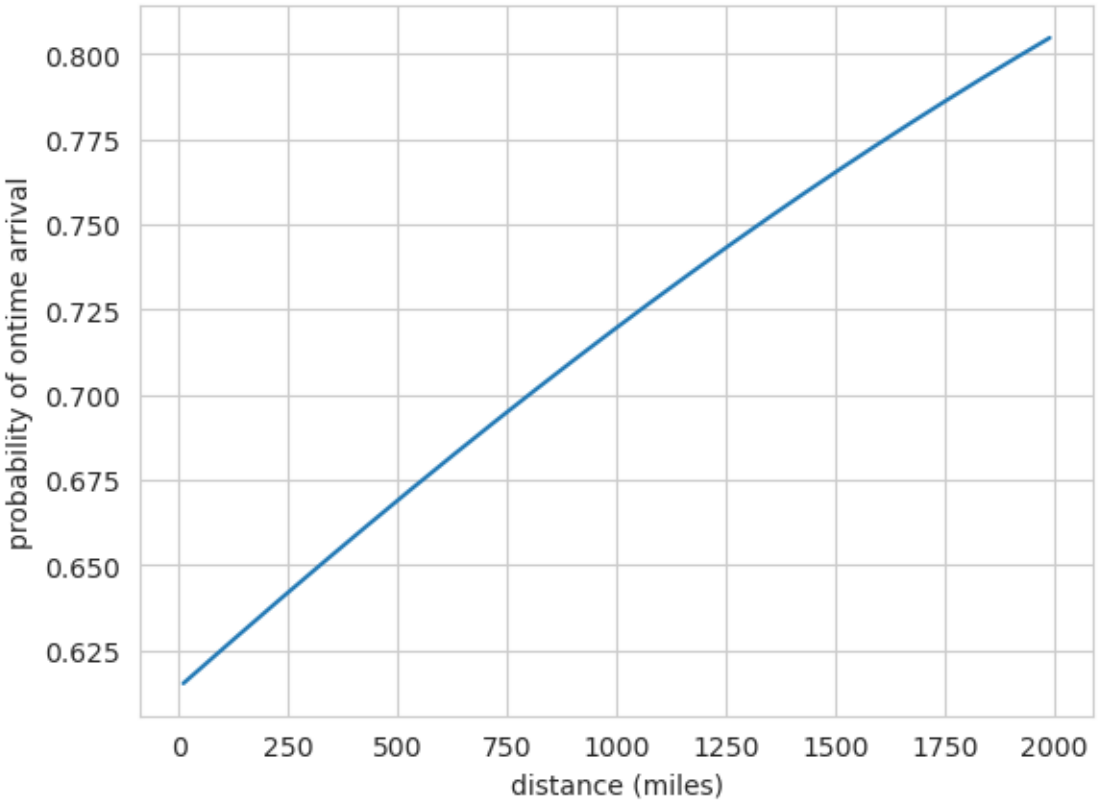
```
0.6689849289476673
```

[37]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# At a departure delay of 20 minutes and a taxi-out time of 10 minutes, this is␣
 ↪how the distance affects the probability that the flight is on time:

dist = np.arange(10, 2000, 10)
prob = [lrmodel.predict([20, 10, d]) for d in dist]
sns.set_style("whitegrid")
ax = plt.plot(dist, prob)
plt.xlabel('distance (miles)')
plt.ylabel('probability of ontime arrival')
```

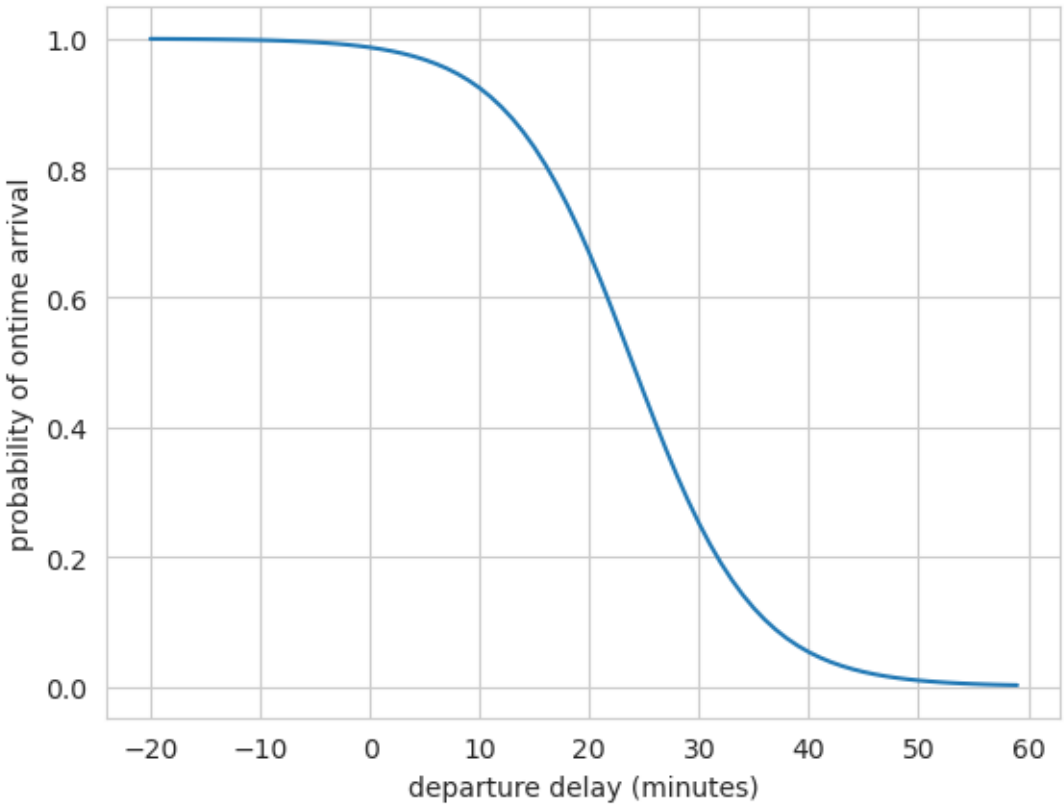
[37]:
```
Text(0, 0.5, 'probability of ontime arrival')
```



the effect is relatively minor. The probability increases from about 0.63 to about 0.76 as the distance changes from a very short hop to a cross-continent flight.

[38]:
```
delay = np.arange(-20, 60, 1)
prob = [lrmodel.predict([d, 10, 500]) for d in delay]
ax = plt.plot(delay, prob)
plt.xlabel('departure delay (minutes)')
plt.ylabel('probability of ontime arrival')
```

[38]:
```
Text(0, 0.5, 'probability of ontime arrival')
```



if you hold the taxi-out time and distance constant and examine the dependence on departure delay, you see a more dramatic impact.

0.9 Evaluate the model

[39]:
```
# Load test data
inputs = 'gs://{}/flights/tzcorr/all_flights-00001-*'.format(BUCKET)
flights = spark.read.json(inputs)
flights.createOrReplaceTempView('flights')
testquery = trainquery.replace("t.is_train_day == 'True'","t.is_train_day ==␣
↪'False'")
```

[40]:
```
# map this training example function to the testing dataset:
testdata = spark.sql(testquery)
examples = testdata.rdd.map(to_example)
```

[41]:
```
# Ask Spark to provide some analysis of the dataset:
testdata.describe().show()
```

```
[Stage 58:==============================>                          (1 + 1) / 2]
```

```
+-------+-----------------+-----------------+-----------------+-------------
---+
|summary|         DEP_DELAY|          TAXI_OUT|         ARR_DELAY|
DISTANCE|
+-------+-----------------+-----------------+-----------------+-------------
---+
|  count|            82184|            82184|            82184|
82184|
|   mean|
8.674377007690062|15.676676725396671|3.8409179402316753|838.9512557188747|
| stddev|38.764341740364586| 8.505730543334973|
41.25995960185183|600.3088554927516|
|    min|            -35.0|              1.0|            -70.0|
1005.00|
|    max|           1576.0|            154.0|           1557.0|
998.00|
+-------+-----------------+-----------------+-----------------+-------------
---+
```

[42]:
```
# Define a eval function and return total cancel, total noncancel, correct␣
↪cancel and correct noncancel flight details:
def eval(labelpred):
    '''
        data = (label, pred)
            data[0] = label
            data[1] = pred
    '''
    cancel = labelpred.filter(lambda data: data[1] < 0.7)
    nocancel = labelpred.filter(lambda data: data[1] >= 0.7)
    corr_cancel = cancel.filter(lambda data: data[0] == int(data[1] >= 0.7)).
↪count()
    corr_nocancel = nocancel.filter(lambda data: data[0] == int(data[1] >= 0.
↪7)).count()

    cancel_denom = cancel.count()
    nocancel_denom = nocancel.count()
    if cancel_denom == 0:
        cancel_denom = 1
    if nocancel_denom == 0:
        nocancel_denom = 1
    return {'total_cancel': cancel.count(), \
            'correct_cancel': float(corr_cancel)/cancel_denom, \
            'total_noncancel': nocancel.count(), \
            'correct_noncancel': float(corr_nocancel)/nocancel_denom \
           }
```

[43]:
```
#  evalute the model by passing correct predicted label:

lrmodel.clearThreshold() # so it returns probabilities
labelpred = examples.map(lambda p: (p.label, lrmodel.predict(p.features)))
print('All flights:')
print(eval(labelpred))
```

```
All flights:
[Stage 66:============================>                            (1 + 1) / 2]
{'total_cancel': 14689, 'correct_cancel': 0.8239498944788617, 'total_noncancel':
67495, 'correct_noncancel': 0.9556411586043411}
```

[44]:
```
# Keep only those examples near the decision threshold which is greater than␣
 ↪65% and less than 75%:
print('Flights near decision threshold:')
labelpred = labelpred.filter(lambda data: data[1] > 0.65 and data[1] < 0.75)
print(eval(labelpred))
```

```
Flights near decision threshold:
[Stage 72:============================>                            (1 + 1) / 2]
{'total_cancel': 714, 'correct_cancel': 0.3711484593837535, 'total_noncancel':
850, 'correct_noncancel': 0.6788235294117647}
```

[ ]:
```

```