Flight delay prediction ii

The prediction accuracy can be improved by using a much bigger data set.

This dataset is 5 GB and contains 52 million flight records. Processing of such a large data set requires use of a Spark cluster.

In [23]:
```
from pyspark.sql import SQLContext
from pyspark.sql.types import *
from pyspark.sql import Row
from pyspark.mllib.regression import LabeledPoint
from pyspark.sql.functions import udf
from pyspark.mllib.linalg import Vectors
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.param import Param, Params
from pyspark.mllib.classification import LogisticRegressionWithLBFG
S, LogisticRegressionModel
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.stat import Statistics
from pyspark.ml.feature import OneHotEncoder, StringIndexer
from pyspark.mllib.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
import sys
import numpy as np
import pandas as pd
import time
import datetime
```

Getting the data and creating the RDD

Size of this dataset is 5 GB,contains nearly 50 million flights.We read data from Swift (Object Storage on Bluemix) into an RDD

In [24]:
```
def set_hadoop_config(credentials):
    """This function sets the Hadoop configuration with given crede
ntials,
    so it is possible to access data using SparkContext"""

    prefix = "fs.swift.service." + credentials['name']
    hconf = sc._jsc.hadoopConfiguration()
    hconf.set(prefix + ".auth.url", credentials['auth_url']+'/v3/au
th/tokens')
    hconf.set(prefix + ".auth.endpoint.prefix", "endpoints")
    hconf.set(prefix + ".tenant", credentials['project_id'])
    hconf.set(prefix + ".username", credentials['user_id'])
    hconf.set(prefix + ".password", credentials['password'])
    hconf.setInt(prefix + ".http.port", 8080)
    hconf.set(prefix + ".region", credentials['region'])
    hconf.setBoolean(prefix + ".public", True)

credentials = {
  'auth_url':'https://identity.open.softlayer.com',
  'project':'object_storage_bcc6ba38_7399_4aed_a47c_e6bcdc959163',
  'project_id':'f26ba12177c44e59adbe243b430b3bf5',
  'region':'dallas',
  'user_id':'bb973e5a84da4fce8c62d95f2e1e5d19',
  'domain_id':'bd9453b2e5e2424388e25677cd26a7cf',
  'domain_name':'1062145',
  'username':'admin_a16bbb9d8d1d051ba505b6e7e76867f61c9d1ac1',
  'password':"""T[{pl6_~9xsjMc8J""",
  'filename':'2001-2008-merged.csv',
  'container':'notebooks',
  'tenantId':'s090-be5845bf9646f1-3ef81b4dcb61'
}
credentials['name'] = 'FlightDelayDemo2'
set_hadoop_config(credentials)

swift_url = "swift://" + credentials['container'] + "." + credentia
ls['name'] + "/" + credentials['filename']
print "Swift URL is %s" % (swift_url)

textFile = sc.textFile(swift_url)
```

```
Swift URL is swift://notebooks.FlightDelayDemo2/2001-2008-merged.c
sv
```

In [27]:
```
textFile.first()
```

Out[27]:
```
u'Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSAr
rTime,UniqueCarrier,FlightNum,TailNum,ActualElapsedTime,CRSElapsed
Time,AirTime,ArrDelay,DepDelay,Origin,Dest,Distance,TaxiIn,TaxiOut
,Cancelled,CancellationCode,Diverted,CarrierDelay,WeatherDelay,NAS
Delay,SecurityDelay,LateAircraftDelay'
```

In [26]:
```
textFileRDD=textFile.map(lambda x: x.split(','))
header = textFileRDD.first()
textRDD = textFileRDD.filter(lambda r: r != header)
```

Creating the Dataframe from RDD

A DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in Python

In [28]:
```
def parse(r):
    try:
        x=Row(Year=int(r[0]),\
          Month=int(r[1]),\
          DayofMonth=int(r[2]),\
          DayOfWeek=int(r[3]),\
          DepTime=int(float(r[4])), \
          CRSDepTime=int(r[5]),\
          ArrTime=int(float(r[6])),\
          CRSArrTime=int(r[7]), \
          UniqueCarrier=r[8],\
          DepDelay=int(float(r[15])),\
          Origin=r[16],\
          Dest=r[17], \
          Distance=int(float(r[18])))
    except:
        x=None
    return x
rowRDD=textRDD.map(lambda r: parse(r)).filter(lambda r:r != None)

airline_df = sqlContext.createDataFrame(rowRDD)
```

In [29]:
```
airline_df=airline_df.withColumn('DepDelayed',airline_df['DepDelay'
]>15)
airline_df.take(4)
```

Out[29]:
```
[Row(ArrTime=1931, CRSArrTime=1934, CRSDepTime=1810, DayOfWeek=3,
DayofMonth=17, DepDelay=-4, DepTime=1806, Dest=u'CLT', Distance=36
1, Month=1, Origin=u'BWI', UniqueCarrier=u'US', Year=2001, DepDela
yed=False),
 Row(ArrTime=1938, CRSArrTime=1934, CRSDepTime=1810, DayOfWeek=4,
DayofMonth=18, DepDelay=-5, DepTime=1805, Dest=u'CLT', Distance=36
1, Month=1, Origin=u'BWI', UniqueCarrier=u'US', Year=2001, DepDela
yed=False),
 Row(ArrTime=1957, CRSArrTime=1934, CRSDepTime=1810, DayOfWeek=5,
DayofMonth=19, DepDelay=11, DepTime=1821, Dest=u'CLT', Distance=36
1, Month=1, Origin=u'BWI', UniqueCarrier=u'US', Year=2001, DepDela
yed=False),
 Row(ArrTime=1944, CRSArrTime=1934, CRSDepTime=1810, DayOfWeek=6,
DayofMonth=20, DepDelay=-3, DepTime=1807, Dest=u'CLT', Distance=36
1, Month=1, Origin=u'BWI', UniqueCarrier=u'US', Year=2001, DepDela
yed=False)]
```

In [30]:
```
def hour_ex(x):
    h=int(str(int(x)).zfill(4)[:2])
    return h
# register as a UDF
f = udf(hour_ex, IntegerType())

#CRSDepTime: scheduled departure time (local, hhmm)
airline_df=airline_df.withColumn('hour', f(airline_df.CRSDepTime))
#airline_df.take(4)
airline_df.registerTempTable("airlineDF")
```

Exploration: Which Airports have the Most Delays?

In [31]:
```
groupedDelay = sqlContext.sql("SELECT Origin, count(*) conFlight,av
g(DepDelay) delay \
                                FROM airlineDF \
                                GROUP BY Origin")

df_origin = groupedDelay.toPandas()
```

In [32]:
```
df_origin.sort('delay',ascending=0).head()
```

```
/usr/local/src/bluemix_jupyter_bundle.v16/notebook/lib/python2.7/s
ite-packages/ipykernel/__main__.py:1: FutureWarning: sort(columns=
....) is deprecated, use sort_values(by=.....)
  if __name__ == '__main__':
```

Out[32]:

	Origin	conFlight	delay
159	FMN	3	203.666667
232	OGD	5	172.400000
313	CYS	2	145.000000
44	BFF	1	131.000000
219	PUB	4	65.500000

Which Routes are typically the most delayed?

In [33]:
```
grp_rout_Delay = sqlContext.sql("SELECT Origin, Dest, count(*) traf
fic,avg(Distance) avgDist,\
                                    avg(DepDelay) avgDelay\
                                FROM airlineDF \
                                GROUP BY Origin,Dest")
rout_Delay = grp_rout_Delay.toPandas()
```

In [35]:
```
rout_Delay.sort('avgDelay',ascending=0).head()
```

```
/usr/local/src/bluemix_jupyter_bundle.v16/notebook/lib/python2.7/s
ite-packages/ipykernel/__main__.py:1: FutureWarning: sort(columns=
....) is deprecated, use sort_values(by=.....)
  if __name__ == '__main__':
```

Out[35]:

	Origin	Dest	traffic	avgDist	avgDelay
5599	CMI	SPI	1	76	587
2169	SUX	OMA	1	80	420
6475	TYS	SDF	1	190	373
1463	BIS	FAR	1	187	369
6158	MCI	SGF	1	159	355

Exploration: Airport Origin delay per month

In [36]:
```
Origin_Airport="SJC"
```

In [37]:
```
df_ORG = sqlContext.sql("SELECT * from airlineDF WHERE origin='"+ O
rigin_Airport+"'")
df_ORG.registerTempTable("df_ORG")
df_ORG.select('ArrTime','CRSArrTime','CRSDepTime',\
              'DayOfWeek','DayofMonth','DepDelay','DepTime','Dest')
.show(2)
```

```
+-------+----------+----------+---------+----------+--------+-----
--+----+
|ArrTime|CRSArrTime|CRSDepTime|DayOfWeek|DayofMonth|DepDelay|DepTi
me|Dest|
+-------+----------+----------+---------+----------+--------+-----
--+----+
|    737|       745|       630|        1|         1|       0|    6
30| SAN|
|    750|       745|       630|        2|         2|       5|    6
35| SAN|
+-------+----------+----------+---------+----------+--------+-----
--+----+
only showing top 2 rows
```

In [38]:
```
print "total flights from this ariport: " + str(df_ORG.count())
```

```
total flights from this ariport: 491408
```

In [39]:
```
grp_carr = sqlContext.sql("SELECT  UniqueCarrier,month, avg(DepDela
y) avgDelay from df_ORG \
                           WHERE DepDelayed=True \
                           GROUP BY UniqueCarrier,month")
s = grp_carr.toPandas()
```

In [40]:
```
ps = s.pivot(index='month', columns='UniqueCarrier', values='avgDel
ay')[['AA','UA','US']]
```

In [44]:
```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams

rcParams['figure.figsize'] = (8,5)
ps.plot(kind='bar', colormap='Greens');
plt.xlabel('Average delay')
plt.ylabel('Month')
plt.title('How much delay does each carrier has in each month?')
```

Out[44]:
```
<matplotlib.text.Text at 0x7f674eafcc90>
```

Exploration: Airport Origin delay per day/hour

In [51]:
```
hour_grouped = df_ORG.filter(df_ORG['DepDelayed']).select('DayOfWee
k','hour','DepDelay').groupby('DayOfWeek','hour').mean('DepDelay')
```

In [52]:
```
rcParams['figure.figsize'] = (10,5)
dh = hour_grouped.toPandas()
c = dh.pivot('DayOfWeek','hour')
X = c.columns.levels[1].values
Y = c.index.values
Z = c.values
plt.xticks(range(0,24), X)
plt.yticks(range(0,7), Y)
plt.xlabel('Hour of Day')
plt.ylabel('Day of Week')
plt.title('Average delay per hours and day?')
plt.imshow(Z)
```

Out[52]:
```
<matplotlib.image.AxesImage at 0x7f674eaf1f10>
```

Modeling: Logistic Regression

build a supervised learning model to predict flight delays for flights leaving SJC

In [11]:
```
Origin_Airport="SJC"
```

In [12]:
```
df_ORG =sqlContext.sql("SELECT * from airlineDF WHERE Origin='"+ Or
igin_Airport+"'")
df_ORG.registerTempTable("df_ORG")
```

Preprocessing: Feature selection

In the next two cell we select the features that we need to create the model.

In [13]:
```
df_model=df_ORG
stringIndexer1 = StringIndexer(inputCol="Origin", outputCol="origin
Index")
model_stringIndexer = stringIndexer1.fit(df_model)
indexedOrigin = model_stringIndexer.transform(df_model)

encoder1 = OneHotEncoder(dropLast=False, inputCol="originIndex", ou
tputCol="originVec")
df_model = encoder1.transform(indexedOrigin)
```

We use labeled point to make local vectors associated with a label/response. In MLlib, labeled points are used in supervised learning algorithms and they are stored as doubles. For binary classification, a label should be either 0 (negative) or 1 (positive).

In [15]:
```
assembler = VectorAssembler(
    inputCols=['Year','Month','DayofMonth','DayOfWeek','hour','DepT
ime','Distance','originVec'],
    outputCol="features")
output = assembler.transform(df_model)
airlineRDD=output.map(lambda row: LabeledPoint([0,1][row['DepDelaye
d']],row['features']))
```

In [11]:
```
output.take(2)
```

Out[11]:
```
[Row(ArrTime=737, CRSArrTime=745, CRSDepTime=630, DayOfWeek=1, Day
ofMonth=1, DepDelay=0, DepTime=630, Dest=u'SAN', Distance=417, Mon
th=1, Origin=u'SJC', UniqueCarrier=u'WN', Year=2001, hour=6, origi
nIndex=0.0, originVec=SparseVector(1, {0: 1.0}), features=DenseVec
tor([2001.0, 1.0, 1.0, 1.0, 6.0, 630.0, 417.0, 1.0])),
 Row(ArrTime=750, CRSArrTime=745, CRSDepTime=630, DayOfWeek=2, Day
ofMonth=2, DepDelay=5, DepTime=635, Dest=u'SAN', Distance=417, Mon
th=1, Origin=u'SJC', UniqueCarrier=u'WN', Year=2001, hour=6, origi
nIndex=0.0, originVec=SparseVector(1, {0: 1.0}), features=DenseVec
tor([2001.0, 1.0, 2.0, 2.0, 6.0, 635.0, 417.0, 1.0]))]
```

In [20]:
```
airlineRDD.take(2)
```

Out[20]:
```
[LabeledPoint(0.0, [2001.0,1.0,1.0,1.0,630.0,417.0,1.0]),
 LabeledPoint(0.0, [2001.0,1.0,2.0,2.0,635.0,417.0,1.0])]
```

In [16]:
```
trainRDD,testRDD=airlineRDD.randomSplit([0.7,0.3])
model = LogisticRegressionWithLBFGS.train(trainRDD)
```

Model Evaluation

In [17]:
```
labelsAndPreds = testRDD.map(lambda p: (p.label, model.predict(p.fe
atures)))
```

In [18]:
```
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / f
loat(testRDD.count())
```

In [19]:
```
print trainErr
```

```
0.140433807839
```

In [33]:
```
labelsAndPreds.take(10)
```

Out[33]:
```
[(0.0, 0),
 (0.0, 0),
 (0.0, 0),
 (0.0, 0),
 (0.0, 0),
 (0.0, 0),
 (0.0, 0),
 (0.0, 0),
 (0.0, 0),
 (0.0, 0)]
```

In [32]:
```
testRDD.take(10)
```

Out[32]:
```
[LabeledPoint(0.0, [2001.0,1.0,16.0,2.0,6.0,630.0,417.0,1.0]),
 LabeledPoint(0.0, [2001.0,1.0,19.0,5.0,6.0,630.0,417.0,1.0]),
 LabeledPoint(0.0, [2001.0,1.0,23.0,2.0,6.0,625.0,417.0,1.0]),
 LabeledPoint(0.0, [2001.0,1.0,31.0,3.0,6.0,625.0,417.0,1.0]),
 LabeledPoint(0.0, [2001.0,1.0,17.0,3.0,21.0,2135.0,417.0,1.0]),
 LabeledPoint(0.0, [2001.0,1.0,3.0,3.0,7.0,755.0,569.0,1.0]),
 LabeledPoint(0.0, [2001.0,1.0,4.0,4.0,7.0,755.0,569.0,1.0]),
 LabeledPoint(0.0, [2001.0,1.0,5.0,5.0,7.0,800.0,569.0,1.0]),
 LabeledPoint(0.0, [2001.0,1.0,9.0,2.0,7.0,755.0,569.0,1.0]),
 LabeledPoint(0.0, [2001.0,1.0,12.0,5.0,7.0,803.0,569.0,1.0])]
```

In [20]:
```
def conf(r):
    if r[0] == r[1] ==1: x= 'TP'
    if r[0] == r[1] ==0: x= 'TN'
    if r[0] == 1 and  r[1] ==0: x= 'FN'
    if r[0] == 0 and  r[1] ==1: x= 'FP'
    return (x)
acc1=labelsAndPreds.map(lambda (v, p): ((v, p),1)).reduceByKey(lamb
da a, b: a + b).take(5)
acc=[(conf(x[0]),x[1]) for x in acc1]
```

In [21]:
```
TP=TN=FP=FN=0.0
for x in acc:
    if x[0]=='TP': TP= x[1]
    if x[0]=='TN': TN= x[1]
    if x[0]=='FP': FP= x[1]
    if x[0]=='FN': FN= x[1]
eps=sys.float_info.epsilon
Accuracy= (TP+TN) / (TP + TN+ FP+FN+eps)
print "Model Accuracy for SJC: %1.2f %%" % (Accuracy*100)
```

```
Model Accuracy for SJC: 85.96 %
```

In [ ]: