Flight delay prediction ¶

In [1]:
```
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 [2]:
```
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 [3]:
```
#remove the header of file
textFileRDD=textFile.map(lambda x: x.split(','))
header = textFileRDD.first()
textRDD = textFileRDD.filter(lambda r: r != header)
```

In [4]:
```
#textFile.take(2)
#textFileRDD.take(2)
```

In [15]:
```
textFileRDD.take(2)
```

Out[15]:
```
[[u'Year',
  u'Month',
```

```
 u'DayofMonth',
 u'DayOfWeek',
 u'DepTime',
 u'CRSDepTime',
 u'ArrTime',
 u'CRSArrTime',
 u'UniqueCarrier',
 u'FlightNum',
 u'TailNum',
 u'ActualElapsedTime',
 u'CRSElapsedTime',
 u'AirTime',
 u'ArrDelay',
 u'DepDelay',
 u'Origin',
 u'Dest',
 u'Distance',
 u'TaxiIn',
 u'TaxiOut',
 u'Cancelled',
 u'CancellationCode',
 u'Diverted',
 u'CarrierDelay',
 u'WeatherDelay',
 u'NASDelay',
 u'SecurityDelay',
 u'LateAircraftDelay'],
[u'2001',
 u'1',
 u'17',
 u'3',
 u'1806',
 u'1810',
 u'1931',
 u'1934',
 u'US',
 u'375',
 u'N700\ufffd\ufffd',
 u'85',
 u'84',
 u'60',
 u'-3',
 u'-4',
 u'BWI',
 u'CLT',
 u'361',
 u'5',
 u'20',
 u'0',
 u'NA',
 u'0',
 u'NA',
 u'NA',
 u'NA',
 u'NA',
 u'NA']]
```

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 [4]:
```
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],\
          FlightNum=int(r[9]),\
          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)
```

add a new column to our data frame to determine the delayed flight against non-delayed ones. Later, we use this column as target/label column in the classification process. So, a binary variable is defined as DepDelayed which its value True for flights having 15 mins or more of delay, and False otherwise.

In [5]:
```
airline_df=airline_df.withColumn('DepDelayed',airline_df['DepDelay'
]>15)
airline_df.take(4)
```

Out[5]:
```
[Row(ArrTime=1931, CRSArrTime=1934, CRSDepTime=1810, DayOfWeek=3,
DayofMonth=17, DepDelay=-4, DepTime=1806, Dest=u'CLT', Distance=36
1, FlightNum=375, Month=1, Origin=u'BWI', UniqueCarrier=u'US', Yea
r=2001, DepDelayed=False),
 Row(ArrTime=1938, CRSArrTime=1934, CRSDepTime=1810, DayOfWeek=4,
DayofMonth=18, DepDelay=-5, DepTime=1805, Dest=u'CLT', Distance=36
1, FlightNum=375, Month=1, Origin=u'BWI', UniqueCarrier=u'US', Yea
r=2001, DepDelayed=False),
 Row(ArrTime=1957, CRSArrTime=1934, CRSDepTime=1810, DayOfWeek=5,
DayofMonth=19, DepDelay=11, DepTime=1821, Dest=u'CLT', Distance=36
1, FlightNum=375, Month=1, Origin=u'BWI', UniqueCarrier=u'US', Yea
r=2001, DepDelayed=False),
 Row(ArrTime=1944, CRSArrTime=1934, CRSDepTime=1810, DayOfWeek=6,
DayofMonth=20, DepDelay=-3, DepTime=1807, Dest=u'CLT', Distance=36
1, FlightNum=375, Month=1, Origin=u'BWI', UniqueCarrier=u'US', Yea
r=2001, DepDelayed=False)]
```

In [6]:
```
# define hour function to obtain hour of day
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")
```

Modeling: Logistic Regression

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

In [7]:
```
Origin_Airport="SJC"
```

In [8]:
```
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 [9]:
```
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 [10]:
```
assembler = VectorAssembler(
    inputCols=['Year','Month','DayofMonth','hour','Distance','origi
nVec'],
    outputCol="features")
output = assembler.transform(df_model)
airlineRDD=output.map(lambda row: LabeledPoint([0,1][row['DepDelaye
d']],row['features']))
```

In [11]:
```
# Spliting dataset into train and test dtasets
trainRDD,testRDD=airlineRDD.randomSplit([0.8,0.2])
model = LogisticRegressionWithLBFGS.train(trainRDD)
```

Model Evaluation

In [13]:
```
# Evaluating the model on testing data
labelsAndPreds = testRDD.map(lambda p: (p.label, model.predict(p.fe
atures)))
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / f
loat(testRDD.count())
```

In [14]:
```
print trainErr
```

```
0.140138601972
```

In [15]:
```
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 [16]:
```
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: 85.99 %
```

In [ ]: