FDA_FAERS Logistic Regression Analysis

Collapse

By centos (http://34.201.57.189.xip.io/centos) — Python 2 Session (Base Image v4) — 18 minutes ago for running

Running


```
from __future__ import print_function
!echo $PYTHON_PATH

import os, sys
from pyspark.sql import *
```

create spark sql session

```
myspark = SparkSession\
    .builder\
    .config("spark.executor.instances", 3 ) \
    .config("spark.executor.memory", "5g") \
    .config("spark.executor.cores", 2) \
    .config("spark.dynamicAllocation.maxExecutors", 10) \
    .config("spark.scheduler.listenerbus.eventqueue.size", 10000) \
    .config("spark.sql.parquet.compression.codec", "snappy") \
    .appName("FDA_FAERS_logisticRegression") \
    .getOrCreate()
```

```
Setting default log level to "ERROR".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use se
tLogLevel(newLevel).
```

```
sc = myspark.sparkContext
import time
print ( time.time())
```

```
1517176360.86
```

```
sc.setLogLevel("ERROR")
print ( myspark )
```

```
<pyspark.sql.session.SparkSession object at 0x7fee3b05c350>
```

make spark print text instead of octal

```
myspark.sql("SET spark.sql.parquet.binaryAsString=true")
```

```
DataFrame[key: string, value: string]
```

read in the data file from HDFS

```
demo = myspark.read.parquet ( "/user/hive/warehouse/medeventsp")
```

print number of rows and type of object

```
print ( demo.count() )
```

```
[Stage 1:==============================>                          (1 +
1) / 2]194463
```

```
demo.show(5)
```

```
+----------+--------+-----------+-------+--------+--------+-----------+-
-------+--------+--------+--------------------+--------+-------+---+----
---+-------+---+-----+----+------+--------+------+--------+------------
---+------------+
|     priid|  caseid|caseversion|i_f_cod|event_dt|  mfr_dt|init_fda_dt|
fda_dt|rept_cod|auth_num|            mfr_num|mfr_sndr|lit_ref|age|age_c
od|age_grp|sex|e_sub|  wt|wt_cod| rept_dt|to_mfr|occp_cod|reporter_count
ry|occr_country|
+----------+--------+-----------+-------+--------+--------+-----------+-
-------+--------+--------+--------------------+--------+-------+---+----
---+-------+---+-----+----+------+--------+------+--------+------------
---+------------+
|1000591924|10005919|         24|      F|20140308|20170823|   20140313|2
0170825|     EXP|        |    PHHY2014CA017890|NOVARTIS|       | 69|
YR|       |  M|    Y|null|      |20170825|      |      OT|
CA|          CA|
| 100066185|10006618|          5|      F|        |20170731|   20140313|2
0170803|     PER|        |US-PFIZER INC-201...|  PFIZER|       | 61|
YR|       |  F|    Y|null|      |20170803|      |      MD|
US|          US|
| 100072049|10007204|          9|      F|20120823|20170629|   20140313|2
0170705|     EXP|        |US-JNJFOC-2014030...| JANSSEN|       | 46|
YR|      A|  M|    Y|null|      |20170705|      |      OT|
US|          US|
|1000745910|10007459|         10|      F|  201607|20170817|   20140313|2
0170829|     EXP|        |US-SANOFI-AVENTIS...| AVENTIS|       | 51|
YR|      A|  M|    Y|null|      |20170829|      |      CN|
US|          US|
|1001243610|10012436|         10|      F|20160419|20170717|   20140314|2
0170724|     EXP|        |US-ACTELION-A-NJ2...|ACTELION|       | 31|
YR|      A|  F|    Y|null|      |20170724|      |      PH|
US|          US|
+----------+--------+-----------+-------+--------+--------+-----------+-
-------+--------+--------+--------------------+--------+-------+---+----
---+-------+---+-----+----+------+--------+------+--------+------------
---+------------+
only showing top 5 rows
```

create a table name to use for sparkSQL queries

```
demo.createOrReplaceTempView("faersdemo")
```

run a query

```
faersagewt=myspark.sql('select age, wt*2.2,sex from faersdemo limit 1000'
```

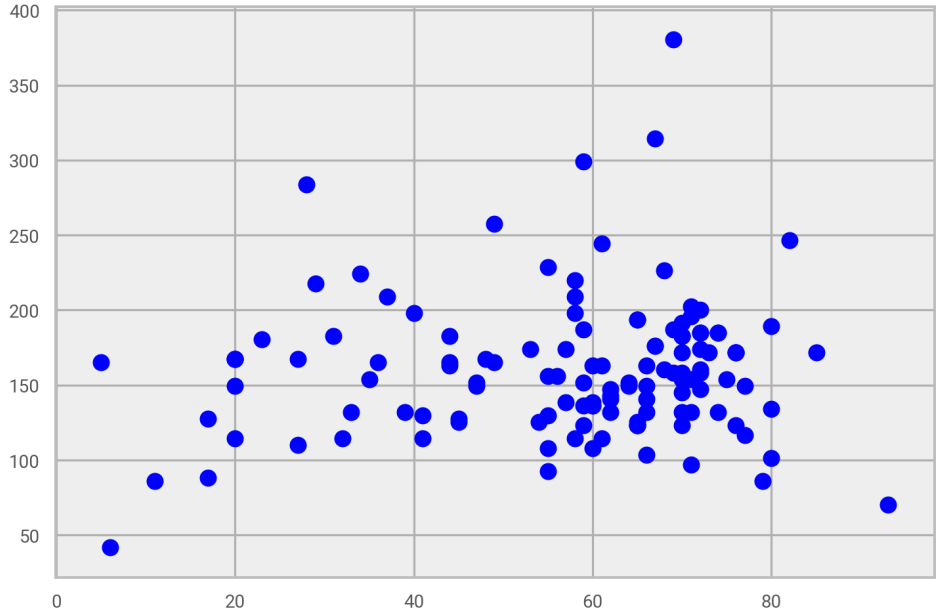
now create pretty graph %matplotlib inline

```
import matplotlib.pyplot as plt
```

use a function so we can call it with varying number of points

```
def plotit(numpts):
```

```
def plotit(numpts):
 for row in faersagewt.take(numpts):
    plt.scatter(row[0],row[1], color=['blue'])
 plt.show()
plotit(400)
```



pairplot to see what we have...

```
import seaborn as sns
import pandas
outcome= myspark.sql('select casecount, mywt, myage, csex, label from log
```

```
outcome.show(3)
```

```
+---------+------------------+-----+----+-----+
|casecount|              mywt|myage|csex|label|
+---------+------------------+-----+----+-----+
|        1|             151.8|   48|   2|    1|
|        2|118.80000000000001|   45|   2|    1|
|        1|             228.8|   51|   2|    1|
+---------+------------------+-----+----+-----+
only showing top 3 rows
```

seaborn wants a pandas dataframe, not a spark dataframe so convert

```
pdsoutcome = outcome.toPandas()
from IPython.display import display
```

display(pdsoutcome)

```
pdsoutcome.dtypes
```

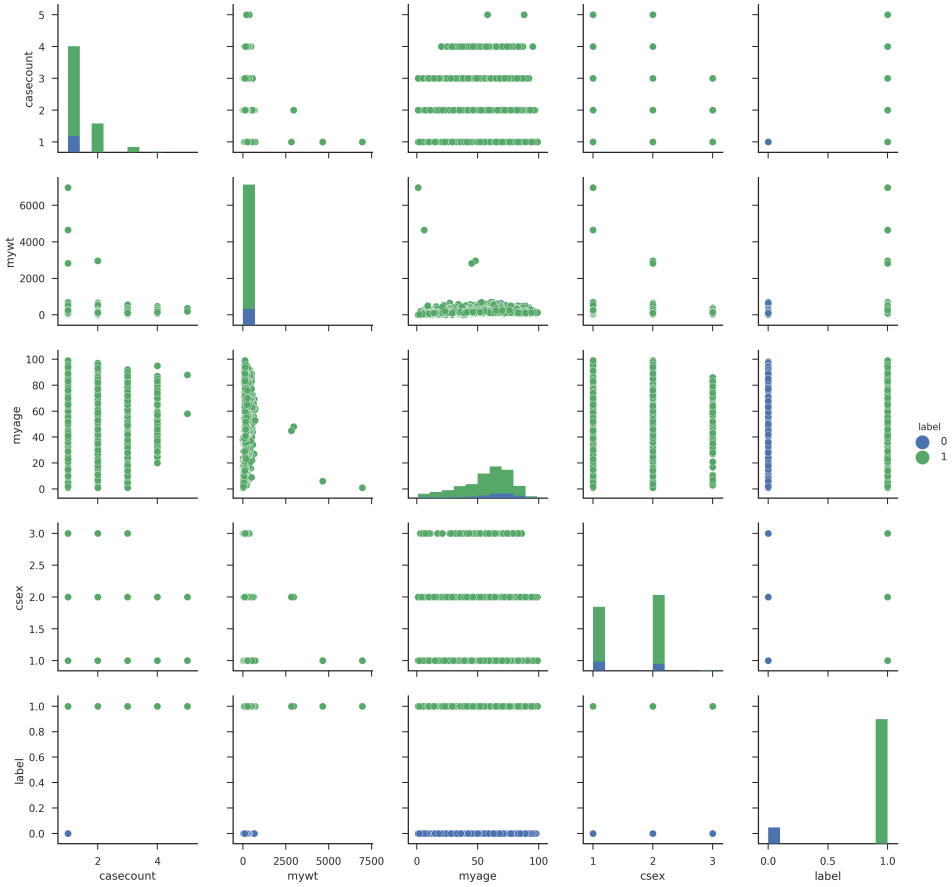
```
casecount      int64
mywt         float64
myage          int64
csex           int64
label          int64
dtype: object
```

```
dtype: object
```

```
sns.set(style="ticks" , color_codes=True)
```

this takes a long time to run: you can see it if you uncomment it

```
g = sns.pairplot(pdsoutcome,  hue="label" )
```



now we need to create a “label” and “features” input for using the sparkML library

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.linalg import Vectors
assembler = VectorAssembler(
    inputCols=[ "casecount", "mywt", "myage", "csex"],
    outputCol="features")
outcomevector = assembler.transform(outcome)
outcomevector.show(2)
```

```
+---------+------------------+-----+----+-----+--------------------+
|casecount|              mywt|myage|csex|label|            features|
+---------+------------------+-----+----+-----+--------------------+
|        1|             151.8|   48|   2|    1|[1.0,151.8,48.0,2.0]|
|        2|118.80000000000001|   45|   2|    1|[2.0,118.80000000...|
+---------+------------------+-----+----+-----+--------------------+
only showing top 2 rows
```

we want to do some data science so split into train and test

```
(train_df, test_df) = outcomevector.randomSplit([0.7, 0.3], seed=1)
train_df.show(2)
```

```
+---------+----+-----+----+-----+------------------+
|casecount|mywt|myage|csex|label|          features|
+---------+----+-----+----+-----+------------------+
|        1| 2.2|   10|   1|    0|[1.0,2.2,10.0,1.0]|
|        1| 4.4|    1|   1|    1| [1.0,4.4,1.0,1.0]|
+---------+----+-----+----+-----+------------------+
only showing top 2 rows
```

```
test_df.show(2)
```

```
+---------+------------------+-----+----+-----+--------------------+
|casecount|             mywt|myage|csex|label|            features|
+---------+------------------+-----+----+-----+--------------------+
|        1|               4.4|    1|   2|    1|   [1.0,4.4,1.0,2.0]|
|        1|6.6000000000000005|    1|   1|    1|[1.0,6.6000000000...|
+---------+------------------+-----+----+-----+--------------------+
only showing top 2 rows
```

if the label field is text need to convert ours is already a float

need to convert from text field to numeric this is a common requirement when using sparkML from pyspark.ml.feature import StringIndexer

this will convert each unique string into a numeric indexer = StringIndexer(inputCol="txtlabel", outputCol="label") indexed = indexer.fit(mydf).transform(mydf) indexed.show(5)

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
```

Fit the model

```
lrModel = lr.fit(train_df)
```

Print the coefficients and intercept for multinomial logistic regression

```
print("Coefficients: \n" + str(lrModel.coefficientMatrix))
```

```
Coefficients:
DenseMatrix([[ 0.,  0.,  0.,  0.]])
```

```
print("Intercept: " + str(lrModel.interceptVector))
```

```
Intercept: [2.0450250259]
```

Extract the summary from the returned LogisticRegressionModel instance trained in the earlier example

```
trainingSummary = lrModel.summary
```

Obtain the objective per iteration

```
objectiveHistory = trainingSummary.objectiveHistory
print("objectiveHistory:")
```

```
objectiveHistory:
```

```
for objective in objectiveHistory:
```

```
print(objective)
```

```
0.355936081112
```

Obtain the receiver-operating characteristic as a dataframe and areaUnderROC.

```
trainingSummary.roc.show()
```

```
+---+---+
|FPR|TPR|
+---+---+
|0.0|0.0|
|1.0|1.0|
|1.0|1.0|
+---+---+
```

```
print("areaUnderROC: " + str(trainingSummary.areaUnderROC))
```

```
areaUnderROC: 0.5
```

Set the model threshold to maximize F-Measure fMeasure = trainingSummary.fMeasureByThreshold maxFMeasure = fMeasure.groupBy().max('F-Measure').select('max(F-Measure)').head() bestThreshold = fMeasure.where(fMeasure['F-Measure'] == maxFMeasure['max(F-Measure)']).select('threshold').head()['threshold'] lr.setThreshold(bestThreshold)

compare and test

Fit the model

```
lrModel = lr.fit(test_df)
```

Print the coefficients and intercept for multinomial logistic regression

```
print("Coefficients: \n" + str(lrModel.coefficientMatrix))
```

```
Coefficients:
DenseMatrix([[ 0.,  0.,  0.,  0.]])
```

```
print("Intercept: " + str(lrModel.interceptVector))
```

```
Intercept: [2.04673800705]
```

Extract the summary from the returned LogisticRegressionModel instance trained in the earlier example

```
trainingSummary = lrModel.summary
```

Obtain the objective per iteration

```
objectiveHistory = trainingSummary.objectiveHistory
print("objectiveHistory:")
```

```
objectiveHistory:
```

```
for objective in objectiveHistory:
        print(objective)
```

```
0.355580838268
```

Obtain the receiver-operating characteristic as a dataframe and areaUnderROC.

```
trainingSummary.roc.show()
```

```
+---+---+
|FPR|TPR|
+---+---+
|0.0|0.0|
```