**Logistic Regression Using Pyspark:--(Binary Classification)**

**Objective:-**

* **To create a logistic model using data, decide best parameters affecting the binary response.**
* **To achieve this we will use LogisticRegressionWithLBFGS from pyspark Mllib.**
* **To select dominant column or significant column we will use Wald’s Statistsics.**
* **Then on basis of accuracy,precision,and RootmeanSquareError(rmse) we finalise the model.**

**Tools Required:-**

1. **Pyspark**
2. **Winscp**
3. **Putty**
4. **Ms excel**

**Data :- we have an insurance data in which (Life) is our Dependent Variable and we have to test its dependency on other 4-columns.**

**Libraries to Import:-**

from pyspark.mllib.classification import LogisticRegressionWithLBFGS, LogisticRegressionModel

from pyspark.mllib.regression import LabeledPoint

from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)

from pyspark.mllib.stat import Statistics

from numpy import array

from pyspark.mllib.evaluation import BinaryClassificationMetrics

from pyspark.mllib.util import MLUtils

from pyspark.sql import Row

from pyspark.ml.classification import LogisticRegression

**Process:-**

After loading the libraries in pyspark shell follow the below steps:-

* Load data to a rdd.
* Split data on basis of delimited operator
* Remove headers if any
* Choose dependent variable as first column and other independent as other columns(all numeric if not do some conversion)
* Define a function that returns labeled point
* Pass the data through the function it will give a labeled point ,which will have one label and its many features
* Now we need to call our Logistic model on our data(we can train the model on 70% data and test it on 30%)
* Calculate weights and store it in a rdd
* Calculate intercept
* Calculate accuracy, std error and RMSE
* Final using wald’s statistic determine the the p value for columns’s weights
* Figure out the significant columns using these values
* And zip the weights with their p-vale
* Save model
* Predict the labels for future data using the model

**CODE:-**

# for loading if hdfs use hdfs:/// before path of file if local use file:/// before path

data = sc.textFile("path")

data = sc.textFile(“file:///home/dms/pawanbi/data.csv”)

# split the data on basis of delimited operator

data1= data.map(lambda x: x.split(","))

# Now we select respective data for 3 models we need on “life”,”mortage”,”finalExpense”

# Loading and Viewing data for different models:-

life\_data = data1.map (lambda x: (x[15],x[13],x[10],x[5],x[2]))

life\_data.first()

(u'Life', u'averagerelationAge', u'annualPayment', u'averageage', u'gender')

mortage\_data = data1.map(lambda x: (x[16],x[13],x[10],x[5],x[2]))

mortage\_data.first()

(u'mortgage', u'averagerelationAge', u'annualPayment', u'averageage', u'gender')

final\_expense\_data = = data1.map(lambda x: (x[17],x[13],x[10],x[5],x[2]))

final\_expense\_data.first()

(u'final\_expense', u'averagerelationAge', u'annualPayment', u'averageage', u'gender')

# Removing Headers from the three files And converting into a labeled point to work with logistic model.

# listing out headers that is a string and the first row of the data

lh1=life\_data.first()

mh1=mortage\_data.first()

fh1=final\_expense\_data.first()

**# removing the headers and storing in a different RDD**

>>> l\_data = life\_data.filter(lambda x: x!=lh1)

>>> m\_data = mortage\_data.filter(lambda x: x!=mh1)

>>> f\_data = final\_expense\_data.filter(lambda x: x!=fh1)

>>> l\_data.first()

(u'1', u'35', u'808.69', u'64', u'1')

>>> m\_data.first()

(u'0', u'35', u'808.69', u'64', u'1')

>>> f\_data.first()

(u'0', u'35', u'808.69', u'64', u'1')

# Now we need to convert them into labeled vectors for model

# So at first we will define a User defined function which can do that:

**# Just copy paste as it is to avoid any indentation errors.Follow the spacing in the code if any error.**

**def parsePoint(line):**

**values = [float(x) for x in line]**

**return LabeledPoint(values[0], values[1:])**

**# Now we will pass our data to the above function to create a labeled RDD, And store them in a different RDD.**

**>>> life1=l\_data.map(parsePoint)**

**>>> mortage1=m\_data.map(parsePoint)**

**>>> expense1=f\_data.map(parsePoint)**

**# Viewing the data:-- ( the first data is the response /label whose dependency will be tested and predicted by our model on the other data which are inside [])**

**>>> life1.first()**

**LabeledPoint(1.0, [35.0,808.69,64.0,1.0])**

**>>> mortage1.first()**

**LabeledPoint(0.0, [35.0,808.69,64.0,1.0])**

**>>> expense1.first()**

**LabeledPoint(0.0, [35.0,808.69,64.0,1.0])**

**# Now we need to create Data frames for these three data to get the probability value.(To view just type data frame name.show() it will list out first 20 rows)**

**lfdf = sqlContext.createDataFrame(life1)**

**mtdf = sqlContext.createDataFrame(mortage1)**

**exdf = sqlContext.createDataFrame(expense1)**

**# For example:-**

**lfdf.show()**

**+--------------------+-----+**

**| features|label|**

**+--------------------+-----+**

**|[35.0,808.69,64.0...| 1.0|**

**|[12.0,574.89,48.0...| 0.0|**

**|[12.0,574.89,48.0...| 0.0|**

**|[12.0,574.89,48.0...| 0.0|**

**|[12.0,574.89,48.0...| 0.0|**

**|[12.0,574.89,48.0...| 0.0|**

**|[12.0,574.89,48.0...| 0.0|**

**|[6.0,290.71,60.0,...| 0.0|**

**|[5.0,274.76,45.0,...| 1.0|**

**|[10.0,276.82,43.0...| 0.0|**

**……………………………………**

**+--------------------+-----+**

**Now we Need to run Model and and see the weights and Intercept for our data we have.**

**# Here we will use two models**

**1.LogisticRegressionModelWithLBFGS(Mllib spark)---runs on rdd**

**2.LogisticRegressionModel(ml spark)—runs on data frames**

**life\_model = LogisticRegressionWithLBFGS.train(life1)**

**mortage\_model = LogisticRegressionWithLBFGS.train(mortage1)**

**expense\_model = LogisticRegressionWithLBFGS.train(expense1)**

**# Now we need to see the coefficients/weights for the independent Variables:-**

**Weights are in order of coefficients take**

**[averagerelationAge, annualPayment, averageage, gender]**

**>>> life\_model.weights**

**DenseVector([-0.0039, -0.0002, -0.0042, -0.192])**

**>>> mortage\_model.weights**

**DenseVector([-0.0056, -0.0001, -0.0038, -0.2047])**

**>>> expense\_model.weights**

**DenseVector([-0.0051, -0.0001, -0.004, -0.2034])**

**# Now we will Accuracy of our model for three datamodels:-**

**>>> life\_labelsAndPreds = life1.map(lambda p: (p.label, life\_model.predict(p.features)))**

**>>> accuracy =life\_labelsAndPreds.filter(lambda (v, p): v == p).count() / float(life1.count())**

**>>> accuracy**

**0.66660666666666668**

**>>>mortage\_labelsAndPreds = mortage1.map(lambda p: (p.label, mortage\_model.predict(p.features)))**

**>>>accuracy =mortage\_labelsAndPreds.filter(lambda (v, p): v == p).count() / float(mortage1.count())**

**>>> accuracy**

**0.66446000000000005**

**>>> expense\_labelsAndPreds = expense1.map(lambda p: (p.label, expense\_model.predict(p.features)))**

**>>> accuracy =expense\_labelsAndPreds.filter(lambda (v, p): v == p).count() / float(expense1.count())**

**>>> accuracy**

**0.66856000000000004**

**# Now we will predict the probability for each row to have define the label correctly or not using ml logistic model as this feature is is yet not available in Mllib logistic model:--**

**# So we need to create three more model using ml library:--**

**lr = LogisticRegression(maxIter=10, regParam=0.01)**

**Here we can define many parameters like regularization,threshold for 1 and 0 but we leave that to default(so don’t run the above code run the below one)**

**lr = LogisticRegression()--- run this**

**lr = LogisticRegression(maxIter=10, regParam=0.01)**

**Now we have to pass the data frame we created to this model for new models:-**

**lr = LogisticRegression()**

**Now we have three model via which we will get the probability of each row.**

**>>> lfdf\_model = lr.fit(lfdf)**

**>>> mtdf\_model = lr.fit(mtdf)**

**>>> exdf\_model = lr.fit(exdf)**

**Now we have three model via which we will get the probability of each row.**

**=**

**>>> life\_para = lfdf\_model.transform(lfdf)**

**>>> life\_para**

**DataFrame[features: vector, label: double, rawPrediction: vector, probability: vector, prediction: double]**

**# we have these many parameters but we need only probability so we will get that only using sql quieries--:**

**life\_prob = life\_para.select("probability")**

**>>> life\_prob.show()**

**>>> life\_prob = life\_para.select("probability")**

**>>> life\_prob.show()**

**+--------------------+**

**| probability|**

**+--------------------+**

**|[0.66361258608104...|**

**|[0.66492181038468...|**

**|[0.66492181038468...|**

**|[0.66492181038468...|**

**# Now we will convert the life\_prob data frame to rdd to zip with original data.**

**life\_prob.rdd.map(tuple).saveAsPickleFile("lifeprob")**

**>>> lp=sc.pickleFile("lifeprob")**

**Similary we will final Probability for our mortage and final expense:-**

**>>> mortage\_para = mtdf\_model.transform(mtdf)**

**>>> mortage\_para**

**DataFrame[features: vector, label: double, rawPrediction: vector, probability: vector, prediction: double]**

**mortage\_prob = mortage\_para.select("probability")**

**>>> mortage\_prob.show()**

**+--------------------+**

**| probability|**

**+--------------------+**

**|[0.66559228017730...|**

**|[0.66523024333357...|**

**|[0.66523024333357...|**

**|[0.66523024333357...|**

**|[0.66523024333357...|**

**|[0.66523024333357...|**

**|[0.66523024333357...|**

**# Now we will convert the mortage\_prob data frame to rdd to zip with original data.**

**mortage\_prob.rdd.map(tuple).saveAsPickleFile("mortageprob")**

**mp=sc.pickleFile("mortageprob")**

**# now finally for the expense:--**

**>>>exdf\_para = exdf\_model.transform(exdf)**

**>>> exdf\_prob - exdf\_para.select("probability")**

**>>> exdf\_prob**

**DataFrame[probability: vector]**

**>>> exdf\_prob.show()**

**+--------------------+**

**| probability|**

**+--------------------+**

**|[0.66955192332755...|**

**|[0.66895499857928...|**

**|[0.66895499857928...|**

**|[0.66895499857928...|**

**|[0.66895499857928...|**

**|[0.66895499857928...|**

**# Now we will convert the expense\_prob data frame to rdd to zip with original data.**

**>>> exdf\_prob.rdd.map(tuple).saveAsPickleFile("expenseprob")**

**>>> ep=sc.pickleFile("expenseprob")**

**Finally we need to add these three probability columns to my original Data:--**

**# But we need little data formatting To work with it Since our lp,mp,ep contains vector and two probabilities each we need to take one only out of them**

**lp1=lp.map(lambda x: (x[0]))**

**lp2 = lp1.map(lambda x: (x[1]))**

**mp1 = mp.map(lambda x: (x[0]))**

**mp2 = mp1.map(lambda x: (x[1]))**

**ep1 = ep.map(lambda x:(x[0]))**

**ep2 = ep1.map(lambda x:(x[1]))**

**data = sc.textFile("file:///home/dms/pawanbi/insurance.csv")**

**>>> data.first()**

**u'fullName,partyKey,gender,birthDate,age,averageage,city,pin,policyKey,policyName,annualPayment,issueDate,relationAge,averagerelationAge,ClsuterID,life,mortgage,finalexpense'**

**>>> h = data.first()**

**>>> data1 = data.filter(lambda x: x!=h)**

**>>> data2= data1.map(lambda x: x.split(","))**

**>>> final\_data = data2.zip(lp).zip(mp).zip(ep)**

**>>> final\_data.first()**

**(((u'Joseph Duncan,2.02E+11,1,4/11/1952,64,64,Boston,2115,1.00E+11,Life Coverage,808.69,10/14/1971,45,35,1,1,0,0', (DenseVector([0.6636, 0.3364]),)), (DenseVector([0.6656, 0.3344]),)), (DenseVector([0.6696, 0.3304]),))**

**# Now we need to do some data Cleaning:--**

**# for that I will use pipe transformation from spark to work with unix..**

**final=final\_data.pipe("sed 's/(((//'") …. To remove brackets**

**final=final.pipe("sed 's/)//'")**

**final=final.pipe("sed 's/)//'")**

**final=final.pipe("sed 's/u//'")**

**# sed has two arguments along with a s as an argument ,**

**/ /**

**1 st argument is what u want to replace**

**2nd with what u want to replace it with**

**If we you see any irregularity in the file just follow the above code and do a find and replace using sed**

**Now I need to save this final output as CSV FILE IN HDFS/LOCAL**

**final.saveAsTextFile(“”Address”)**

**File will be saved in parts**