# CISC 5950 - Big Data Programming

# Project 2 - Part 2 Report


**Team Members:**

**Revathi Bhuvaneswari**

**Tianying Luo**

**Youfei Zhang**

**Yue Zeng**


**Spring 2019**

# P2: Heart Disease Prediction using Logistic Regression

**Bash script:** Import the data file from HDFS.

```
#!/bin/bash
#../../start.sh
source ../../env.sh
/usr/local/hadoop/bin/hdfs dfs -rm -r /part2/input/
/usr/local/hadoop/bin/hdfs dfs -rm -r /part2/output/
/usr/local/hadoop/bin/hdfs dfs -mkdir -p /part2/input/
/usr/local/hadoop/bin/hdfs dfs -copyFromLocal ../../data/framingham.csv /part2/input/
/usr/local/spark/bin/spark-submit --master=spark://$SPARK_MASTER:7077 ./framingham.py hdfs://$SPARK_MASTER:9000/part2/input/
#../../stop.sh
```

**Python Script:**

**Stage 1:** The train / test data are read from the CSV files and loaded into dataframes.

```
reload(sys)
sys.setdefaultencoding('utf8')
if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: Q2.py <directory_of_files>", file=sys.stderr)
        sys.exit(-1)

    spark = SparkSession.builder.appName('lr-predic').getOrCreate()
    df = spark.read.csv(sys.argv[1], header = True, inferSchema = True)
    df.show()
    cols=df.columns
    df.printSchema()
```

**Stage 2:** The dataset is cleaned, the vectors of ‘features’ & ‘label’  are added to the dataframe.

```
#Filter out all the 'NA' values
df = df.filter((df.cigsPerDay != 'NA')& (df.BPMeds != 'NA') & (df.totChol != 'NA') & (df.BMI != 'NA') & (df.heartRate != 'NA') & (df.glucose != 'NA'))
string_cols=['cigsPerDay', 'BPMeds','totChol','BMI', 'heartRate', 'glucose']
for col_name in string_cols:
    df = df.withColumn(col_name, col(col_name).cast('float'))

#Count of the class values
df.groupby('TenYearCHD').count().show()

#Drop the column of 'education'
#Generative statistics conclusion
cols.remove('education')
cols.insert(0, 'Summary')
df.describe().select(cols[:len(cols)//2]).show()
df.describe().select(cols[len(cols)//2:-1]).show()

#Use pipeline to combine all the features in one single feature vector
cols.remove('TenYearCHD')
cols.remove('Summary')
print(cols)

#['male', 'age', 'currentSmoker', 'cigsPerDay', 'BPMeds', 'prevalentStroke', 'prevalentHyp',
# 'diabetes', 'totChol', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose']

df = ohe(df)
df.printSchema()
```

The cleaning process is done by first filtering out all the ‘NA’ values from the following columns: ‘cigsPerDay’, ‘BPMeds’, ‘totChol’, ‘BMI’, ‘heartRate’, ‘glucose'. ‘NA’ values represent missing values in the dataset and are identified as string value. Then all the columns of string type are altered by converting them into float type. Additionally, the unwanted column of ‘education’ is dropped.

The following function is used to one-hot-encode all categorical columns, and get vector representation of all features. String Indexer and One Hot Encoder Estimator are used for this.

```python
# converts sparse vector to dense vector
def sparse_to_array(v):
    v = DenseVector(v)
    new_array = list([float(x) for x in v])
    return new_array

# returns one-hot-encoded stages
def ohe(df):
    # One-hot-encode category columns
    categoricalColumns = ['education','currentSmoker', 'BPMeds', 'prevalentStroke', 'prevalentHyp', 'diabetes']
    stages = []
    for categoricalCol in categoricalColumns:
        stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol = categoricalCol + 'Index')
        encoder = OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
        stages += [stringIndexer, encoder]

    # vectorize numeric columns
    numericCols = ['male', 'age', 'cigsPerDay', 'totChol', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose']
    assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
    assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
    stages += [assembler]

    # perform final pipeline to get vectorized feature column
    pipeline = Pipeline().setStages(stages)
    pipelineModel = pipeline.fit(df)
    df = pipelineModel.transform(df)
    selectedCols = ['TenYearCHD', 'features'] + cols
    df = df.select(selectedCols)

    # convert sparse vector initial_feature column to dense vector
    sparse_to_array_udf = udf(sparse_to_array, T.ArrayType(T.FloatType()))
    df = df.withColumn('dense_vector_features', sparse_to_array_udf('features'))
    ud_f = udf(lambda r : Vectors.dense(r), VectorUDT())
    df = df.withColumn('features', ud_f('dense_vector_features'))
    return df
```

**Stage 3:** In this stage, a new column of 'Scaled_features' is created by using StandardScaler to standardize the newly feature values according to their different calibration. Also, since the classification values are imbalanced. Balancing ratio is calculated, based on which class weights will be calculated and applied to different classes. In addition, ChiSqSelector method is used to select the feature with p-value below 0.05, which proves a strong predictive power. At last, use the class column as labelCol, 'Aspect' (generated by chi-square feature selection) as feautresCol, and classweights as weightCol, and the iteration time is set to be 10. After that, split the dataset into train and test by 80/20. Train and fit the model by using the training dataset.

```python
#Standarization
standardscaler=StandardScaler().setInputCol("features").setOutputCol("Scaled_features")
df=standardscaler.fit(df).transform(df)
df.select("features","Scaled_features").show(5)

#Split train & test
train, test = df.randomSplit([0.8, 0.2], seed=123)

total=float(train.select("TenYearCHD").count())
numPositives=train.select("TenYearCHD").where('TenYearCHD == 1').count()
per_ones=(float(numPositives)/float(total))*100
numNegatives=float(total-numPositives)
print('\n\nThe number of Class 1 are {}'.format(numPositives))
print('\n\nPercentage of Class 1 are {}'.format(per_ones))

#Add balancing ratio
BalancingRatio= numNegatives/total
print('\n\nBalancingRatio = {}'.format(BalancingRatio))

#Creat a new column named "classWeights" in the "train" dataset
train=train.withColumn("classWeights", when(train.TenYearCHD==1, BalancingRatio).otherwise(1-BalancingRatio))
train.select("classWeights", "TenYearCHD").show(10)

#Feature selection
css = ChiSqSelector(featuresCol='features',outputCol='Aspect',labelCol='TenYearCHD',fpr=0.05)
train=css.fit(train).transform(train)
test=css.fit(test).transform(test)
test.select("Aspect").show(5,truncate=False)

#Train the lr model
lr = LogisticRegression(labelCol="TenYearCHD", featuresCol="Aspect",weightCol="classWeights", maxIter=10)
model=lr.fit(train)
```

**Stage 4:** In this last stage, the trained Logistic Regression model is used to make predictions by applying the transform() function on both the train as well as the test data frames. Then extract the counts of TP, TN, FP, and FN by filtering the corresponding pairs and counting them. The Area Under the ROC is calculated using the Binary Classification Evaluator by passing in the actual predictions data frame, which has both the labels and predictions.

```
predict_train=model.transform(train)
predict_test=model.transform(test)
predict_train.select("TenYearCHD","prediction").show(10)
predict_test.select("TenYearCHD","prediction").show(10)

#Evaluation of the train model
TP_train = predict_train.filter((predict_train.TenYearCHD == 1) & (predict_train.prediction == 1.0)).count()
TN_train = predict_train.filter((predict_train.TenYearCHD == 0) & (predict_train.prediction == 0.0)).count()
FP_train = predict_train.filter((predict_train.TenYearCHD == 0) & (predict_train.prediction == 1.0)).count()
FN_train = predict_train.filter((predict_train.TenYearCHD == 1) & (predict_train.prediction == 0.0)).count()
evaluation(TP_train, TN_train, FP_train, FN_train)

#Evaluation of the test model
TP_test = predict_test.filter((predict_test.TenYearCHD == 1) & (predict_test.prediction == 1.0)).count()
TN_test = predict_test.filter((predict_test.TenYearCHD == 0) & (predict_test.prediction == 0.0)).count()
FP_test = predict_test.filter((predict_test.TenYearCHD == 0) & (predict_test.prediction == 1.0)).count()
FN_test = predict_test.filter((predict_test.TenYearCHD == 1) & (predict_test.prediction == 0.0)).count()
evaluation(TP_test, TN_test, FP_test, FN_test)


#ROC for train & test
evaluator=BinaryClassificationEvaluator(rawPredictionCol="rawPrediction",labelCol="TenYearCHD")
predict_test.select("TenYearCHD","rawPrediction","prediction","probability").show(5, truncate = False)

print("\n\nThe area under ROC for train set is {}".format(evaluator.evaluate(predict_train)))
print("\n\nThe area under ROC for test set is {}".format(evaluator.evaluate(predict_test)))
```

The following function is used to calculate and output the accuracy, misclassification rate, sensitivity, specificity, positive/negative predictive value and positive/negative likelihood ratio.

```
def evaluation(TP, TN, FP, FN):
    print('\n\n', 'The confusion matrix is: \n\n', '[[',TN,FP,']\n\n','[',FN,TP,']]', '\n\n')
    #Model Evaluation - Statistics
    sensitivity=TP/float(TP+FN)
    specificity=TN/float(TN+FP)

    print('\n\nThe acuuracy of the model = (TP+TN)/(TP+TN+FP+FN) = ',(TP+TN)/float(TP+TN+FP+FN),'\n\n',

    'The Missclassification = 1-Accuracy = ',1-((TP+TN)/float(TP+TN+FP+FN)),'\n\n',

    'Sensitivity or True Positive Rate = TP/(TP+FN) = ',TP/float(TP+FN),'\n\n',

    'Specificity or True Negative Rate = TN/(TN+FP) = ',TN/float(TN+FP),'\n\n',

    'Positive Predictive value = TP/(TP+FP) = ',TP/float(TP+FP),'\n\n',

    'Negative predictive Value = TN/(TN+FN) = ',TN/float(TN+FN),'\n\n',

    'Positive Likelihood Ratio = Sensitivity/(1-Specificity) = ',sensitivity/(1-specificity),'\n\n',

    'Negative likelihood Ratio = (1-Sensitivity)/Specificity = ',(1-sensitivity)/specificity,'\n\n')
```

## Output:

| male | age | education | currentSmoker | cigsPerDay | BPMeds | prevalentStroke | prevalentHyp | diabetes | totChol | sysBP | diaBP | BMI | heartRate | glucose | TenYearCHD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 39 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 195 | 106.0 | 70.0 | 26.97 | 80 | 77 | 0 |
| 0 | 46 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 250 | 121.0 | 81.0 | 28.73 | 95 | 76 | 0 |
| 1 | 48 | 1 | 1 | 20 | 0 | 0 | 0 | 0 | 245 | 127.5 | 80.0 | 25.34 | 75 | 70 | 0 |
| 0 | 61 | 3 | 1 | 30 | 0 | 0 | 1 | 0 | 225 | 150.0 | 95.0 | 28.58 | 65 | 103 | 1 |
| 0 | 46 | 3 | 1 | 23 | 0 | 0 | 0 | 0 | 285 | 130.0 | 84.0 | 23.1 | 85 | 85 | 0 |
| 0 | 43 | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 228 | 180.0 | 110.0 | 30.3 | 77 | 99 | 0 |
| 0 | 63 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 205 | 138.0 | 71.0 | 33.11 | 60 | 85 | 1 |
| 0 | 45 | 2 | 1 | 20 | 0 | 0 | 0 | 0 | 313 | 100.0 | 71.0 | 21.68 | 79 | 78 | 0 |
| 1 | 52 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 260 | 141.5 | 89.0 | 26.36 | 76 | 79 | 0 |
| 1 | 43 | 1 | 1 | 30 | 0 | 0 | 1 | 0 | 225 | 162.0 | 107.0 | 23.61 | 93 | 88 | 0 |
| 0 | 50 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 254 | 133.0 | 76.0 | 22.91 | 75 | 76 | 0 |
| 0 | 43 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 247 | 131.0 | 88.0 | 27.64 | 72 | 61 | 0 |
| 1 | 46 | 1 | 1 | 15 | 0 | 0 | 1 | 0 | 294 | 142.0 | 94.0 | 26.31 | 98 | 64 | 0 |
| 0 | 41 | 3 | 0 | 0 | 1 | 0 | 1 | 0 | 332 | 124.0 | 88.0 | 31.31 | 65 | 84 | 0 |
| 0 | 39 | 2 | 1 | 9 | 0 | 0 | 0 | 0 | 226 | 114.0 | 64.0 | 22.35 | 85 | NA | 0 |
| 0 | 38 | 2 | 1 | 20 | 0 | 0 | 1 | 0 | 221 | 140.0 | 90.0 | 21.35 | 95 | 70 | 1 |
| 1 | 48 | 3 | 1 | 10 | 0 | 0 | 1 | 0 | 232 | 138.0 | 90.0 | 22.37 | 64 | 72 | 0 |
| 0 | 46 | 2 | 1 | 20 | 0 | 0 | 0 | 0 | 291 | 112.0 | 78.0 | 23.38 | 80 | 89 | 1 |
| 0 | 38 | 2 | 1 | 5 | 0 | 0 | 0 | 0 | 195 | 122.0 | 84.5 | 23.24 | 75 | 78 | 0 |
| 1 | 41 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 195 | 139.0 | 88.0 | 26.88 | 85 | 65 | 0 |

***Figure 1: Initial DataFrame (first 20 rows)***

```
root
 |-- male: integer (nullable = true)
 |-- age: integer (nullable = true)
 |-- education: string (nullable = true)
 |-- currentSmoker: integer (nullable = true)
 |-- cigsPerDay: string (nullable = true)
 |-- BPMeds: string (nullable = true)
 |-- prevalentStroke: integer (nullable = true)
 |-- prevalentHyp: integer (nullable = true)
 |-- diabetes: integer (nullable = true)
 |-- totChol: string (nullable = true)
 |-- sysBP: double (nullable = true)
 |-- diaBP: double (nullable = true)
 |-- BMI: string (nullable = true)
 |-- heartRate: string (nullable = true)
 |-- glucose: string (nullable = true)
 |-- TenYearCHD: integer (nullable = true)
```

***Figure 2: Schema of the Initial DataFrame***

| TenYearCHD | count |
|---|---|
| 1 | 644 |
| 0 | 3596 |

| TenYearCHD | count |
|---|---|
| 1 | 572 |
| 0 | 3179 |

***Figure 3: Classification Counts Before & After Data Cleaning***

| Summary | male | age | currentSmoker | cigsPerDay | BPMeds | prevalentStroke | prevalentHyp |
|---|---|---|---|---|---|---|---|
| count | 3751 | 3751 | 3751 | 3751 | 3751 | 3751 | 3751 |
| mean | 0.4452146094374833 | 49.57344708077846 | 0.48840309250866437 | 9.008531058384431 | 0.03039189549453479 | 0.005598507064782724 | 0.3119168221807518 |
| stddev | 0.49705575870704916 | 8.570204436756383 | 0.49993213793450775 | 11.925096527979754 | 0.17168601101128225 | 0.07462337676919495 | 0.4633378373062201 |
| min | 0 | 32 | 0 | 0.0 | 0.0 | 0 | 0 |
| max | 1 | 70 | 1 | 70.0 | 1.0 | 1 | 1 |

| diabetes | totChol | sysBP | diaBP | BMI | heartRate | glucose |
|---|---|---|---|---|---|---|
| 3751 | 3751 | 3751 | 3751 | 3751 | 3751 | 3751 |
| 0.027192748600373233 | 236.9280191948136 | 132.3684350839776 | 82.93854972007465 | 25.80828845268947 | 75.70407891229006 | 81.88003199146894 |
| 0.1626663986140648 | 44.61159360579599 | 22.04652246327564 | 11.932779343310061 | 4.06559867696019 | 11.956382146147071 | 23.882233365111478 |
| 0 | 113.0 | 83.5 | 48.0 | 15.54 | 44.0 | 40.0 |
| 1 | 696.0 | 295.0 | 142.5 | 56.8 | 143.0 | 394.0 |

*Figure 4: Generative Statistics Analysis of Each Attribute*

```
root
 |-- TenYearCHD: integer (nullable = true)
 |-- features: vector (nullable = true)
 |-- male: integer (nullable = true)
 |-- age: integer (nullable = true)
 |-- currentSmoker: integer (nullable = true)
 |-- cigsPerDay: float (nullable = true)
 |-- BPMeds: float (nullable = true)
 |-- prevalentStroke: integer (nullable = true)
 |-- prevalentHyp: integer (nullable = true)
 |-- diabetes: integer (nullable = true)
 |-- totChol: float (nullable = true)
 |-- sysBP: double (nullable = true)
 |-- diaBP: double (nullable = true)
 |-- BMI: float (nullable = true)
 |-- heartRate: float (nullable = true)
 |-- glucose: float (nullable = true)
 |-- dense_vector_features: array (nullable = true)
 |    |-- element: float (containsNull = true)
```

*Figure 5: Schema of the DataFrame After Data Cleaning*

```
+--------------------+--------------------+
|            features|     Scaled_features|
+--------------------+--------------------+
|[0.0,0.0,0.0,1.0,...|[0.0,0.0,0.0,3.16...|
|[0.0,1.0,0.0,0.0,...|[0.0,2.1956978139...|
|[1.0,0.0,0.0,0.0,...|[2.03538650844691...|
|[0.0,0.0,1.0,0.0,...|[0.0,0.0,2.713096...|
|[0.0,0.0,1.0,0.0,...|[0.0,0.0,2.713096...|
+--------------------+--------------------+
only showing top 5 rows
```

*Figure 6: Scaled_features of Standardization*

```
The number of Class 1 are 460


Percentage of Class 1 are 15.343562374916612


BalancingRatio = 0.8465643762508339
```

*Figure 7: Percentage of Class 1 & Balancing Ratio*

```
+------------------+----------+
|      classWeights|TenYearCHD|
+------------------+----------+
|0.15343562374916608|         0|
|0.15343562374916608|         0|
|0.15343562374916608|         0|
|0.15343562374916608|         0|
|0.15343562374916608|         0|
|0.15343562374916608|         0|
|0.15343562374916608|         0|
|0.15343562374916608|         0|
|0.15343562374916608|         0|
|0.15343562374916608|         0|
+------------------+----------+
only showing top 10 rows
```

```
+------------------------------------------------------------------------------------------------------+
|Aspect                                                                                                |
+------------------------------------------------------------------------------------------------------+
|[0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,1.0,0.0,50.0,15.0,253.0,132.0,84.5,27.959999084472656,88.0,73.0] |
|[0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,1.0,0.0,51.0,15.0,240.0,180.0,107.5,25.329999923706055,68.0,80.0]|
|[0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,1.0,0.0,57.0,20.0,246.0,160.0,92.5,30.739999771118164,84.0,76.0] |
|[0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,1.0,1.0,67.0,25.0,221.0,144.0,84.0,24.920000076293945,72.0,73.0] |
|[0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,0.0,37.0,20.0,223.0,115.0,72.0,22.709999084472656,76.0,63.0] |
+------------------------------------------------------------------------------------------------------+
only showing top 5 rows
```

*Figure 8: Classweights & Aspect*

| TenYearCHD | prediction |
|---|---|
| 0 | 1.0 |
| 0 | 1.0 |
| 0 | 1.0 |
| 0 | 1.0 |
| 0 | 1.0 |
| 0 | 1.0 |
| 0 | 0.0 |
| 0 | 0.0 |
| 0 | 0.0 |
| 0 | 0.0 |

only showing top 10 rows

| TenYearCHD | prediction |
|---|---|
| 0 | 0.0 |
| 0 | 1.0 |
| 0 | 1.0 |
| 0 | 1.0 |
| 0 | 0.0 |
| 0 | 0.0 |
| 0 | 0.0 |
| 0 | 0.0 |
| 0 | 0.0 |
| 0 | 1.0 |

only showing top 10 rows

***Figure 9: Train & Test Prediction Outputs***

| TenYearCHD | rawPrediction | prediction | probability |
|---|---|---|---|
| 0 | [-0.1567527896942406,0.1567527896942406] | 1.0 | [0.46089184842530045,0.5391081515746997] |
| 0 | [-1.79364229862992,1.79364229862992] | 1.0 | [0.14262674731316835,0.8573732526868316] |
| 0 | [-1.616040225550969,1.616040225550969] | 1.0 | [0.16575169569904596,0.8342483043009541] |
| 0 | [0.9793847307922181,-0.9793847307922181] | 0.0 | [0.7269861164242096,0.2730138835757904] |
| 0 | [0.6310721597565216,-0.6310721597565216] | 0.0 | [0.6527325313989591,0.34726746860104085] |

only showing top 5 rows

***Figure 10: Probability of Predictions***

```
The area under ROC for train set is 0.7269039999999907
```

```
Evaluation Results for Training Dataset


 The confusion matrix is:

 [[ 1677 823 ]

 [ 154 296 ]]


The acuuracy of the model = (TP+TN)/(TP+TN+FP+FN) =  0.6688135593220339

 The Missclassification = 1-Accuracy =  0.3311864406779661

 Sensitivity or True Positive Rate = TP/(TP+FN) =  0.6577777777777778

 Specificity or True Negative Rate = TN/(TN+FP) =  0.6708

 Positive Predictive value = TP/(TP+FP) =  0.2645218945487042

 Negative predictive Value = TN/(TN+FN) =  0.9158929546695794

 Positive Likelihood Ratio = Sensitivity/(1-Specificity) =  1.9981098960442822

 Negative likelihood Ratio = (1-Sensitivity)/Specificity =  0.5101702776121381
```

***Figure 11: Evaluation Results for Training Dataset***

```
The area under ROC for test set is 0.774487554021107
```

```
Evaluation Results for Test Dataset


 The confusion matrix is:

 [[ 482 197 ]

 [ 33 89 ]]


The acuuracy of the model = (TP+TN)/(TP+TN+FP+FN) =  0.7128589263420724

 The Missclassification = 1-Accuracy =  0.2871410736579276

 Sensitivity or True Positive Rate = TP/(TP+FN) =  0.7295081967213115

 Specificity or True Negative Rate = TN/(TN+FP) =  0.7098674521354934

 Positive Predictive value = TP/(TP+FP) =  0.3111888111888112

 Negative predictive Value = TN/(TN+FN) =  0.9359223300970874

 Positive Likelihood Ratio = Sensitivity/(1-Specificity) =  2.514396271948074

 Negative likelihood Ratio = (1-Sensitivity)/Specificity =  0.38104550710835994
```

*Figure 12: Evaluation Results for Testing Dataset*