**Pima Indian Diabetes Prediction**

**Import basic Libraries**

**import** pandas **as** pd *# panda is a dataframe library*

**import** matplotlib.pyplot **as** plt *# matplotlib.pyplot plot data*

**import** numpy **as** np *# numpy provide n - dimension object support*

​

*# plot data on the same line instead of different window.*

**%**matplotlib inline

---------------------------------------------------------------------------------------------------------------

**Import CSV File and Review**

pd.read is a function from Pandas package to read the dataset. By default it consider the first row of the dataset as the header. Before reading the dataset you must upload it here.

df = pd.read\_csv("/home/nbuser/library/pima-data.csv")

df.shape

**# head is a method used to display dataset from the top**

df.head(10)

**# tail is a method used to display dataset from the bottom**

df.tail(10)

==================================================================

**Pre-processing of Dataset**

**We will check for Columns**

***not used***

***no values***

***duplicates***

**Correlated columns doesn’t carry useful information and often confuse Algorithm. Also, it bias the record because algorithm treat each column as independent. So its better to drop correlated columns.**

**Check for Null value**

isnull().values.any() is a method used to find null values in the dataset.

df.isnull().values.any()

**def** plot\_corr(df, size=10):

"""

Function plots a graphical correlation matrix for each pair of columns in the dataframe.

​

Input:

df: pandas DataFrame

size: vertical and horizontal size of the plot

​

Displays:

matrix of correlation between columns. Blue-cyan-yellow-red-darkred => less to more correlated

0 ------------------> 1

Expect a darkred line running from top left to bottom right

"""

​

corr = df.corr() *# data frame correlation function*

fig, ax = plt.subplots(figsize=(size, size)) *# ax is used to hold the corelation value.*

ax.matshow(corr) *# color code the rectangles by correlation value*

plt.xticks(range(len(corr.columns)), corr.columns) *# draw x tick marks*

plt.yticks(range(len(corr.columns)), corr.columns) *# draw y tick marks*

​

plot\_corr(df)

df.head(5) *# check 1:1 correaltion bwtween skin and thinkness columns*

**Delete skin column**

**del** df['skin']

df.head(5)

plot\_corr(df)

**Check Data Type**

df.head(10) **change True to 1 and False to 0**

define a map dictinary that True is 1 and False is 0.

diabetes\_map = {**True** : 1, **False** : 0}

df['diabetes'] = df['diabetes'].map(diabetes\_map)

df.head(5)

**Check True / False ratio**

num\_true = len(df.loc[df['diabetes'] == 1])

num\_false = len(df.loc[df['diabetes'] == 0])

print("Number of True cases: {0} ({1}%)".format(num\_true, (num\_true**/** (num\_true **+** num\_false)) **\*** 100))

print("Number of False cases: {0} ({1}%)".format(num\_false, (num\_false**/** (num\_true **+** num\_false)) **\*** 100))

Number of True cases: 268 (34.89583333333333%)

Number of False cases: 500 (65.10416666666666%)

**With these many no of True and False cases, we can employ standard ML algorithm.**

**Selection of Algorithm**

Algorithm is an engine that drives the process. Using the Algorithm logic, data are analysed. Over 50 standard in Sci-kit learn package.

**Alogrithm Decision Factor**

1. Learning Type (Supervised and Unsupervised) - 28
2. Result (Regression and Classification) - 20
3. Complexity (simple and ensamble (complex))
4. Baisc Vs Enhanced

We Will look into 3 Candidate Algorithm

**1. Naive base**

**It is based on Bayes Theorem.**

**It checks how often the feature correates with the lable.**

**Every features are considered independent and treated equally.**

**It is easy to understand and is fast.**

**Stable ( with small change in training data, the output doesnt vary much )**

**2. Logistic Regression**

**finds the relationship between each features and assign values based on their impact on predicting lable.**

**3. Decesion Tree**

**Binary Tree**

**Node containing decision**

**Require enough amount of data to determine nodes and split.**

**Training the Model**

**Sci kit learn Library**

Designed to work with Pandas, NumPy, SciPy

Toolset for Training and Evaluation of Dataset

***Data Splitting***

***Pre processing***

***Feature Selection***

***Model Training***

***Model Tunning***

**Splitting the Dataset**

70 % for training and 30 % for testing

**from** sklearn.cross\_validation **import** train\_test\_split

​

feature\_col\_names = ['num\_preg', 'glucose\_conc', 'diastolic\_bp', 'thickness', 'insulin', 'bmi', 'diab\_pred', 'age']

predicted\_class\_names = ['diabetes']

​

X = df[feature\_col\_names].values *# predictor feature columns (8 X m)*

y = df[predicted\_class\_names].values *# predicted class (1=true, 0=false) column (1 X m)*

split\_test\_size = 0.30

​

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=split\_test\_size, random\_state=42)

*# test\_size = 0.3 is 30%, 42 is the answer to everything*

print("{0:0.2f}% in training set".format((len(X\_train)**/**len(df.index)) **\*** 100))

print("{0:0.2f}% in test set".format((len(X\_test)**/**len(df.index)) **\*** 100))

69.92% in training set

30.08% in test set

**Splitting is properly done**

print("Original True : {0} ({1:0.2f}%)".format(len(df.loc[df['diabetes'] == 1]), (len(df.loc[df['diabetes'] == 1])**/**len(df.index)) **\*** 100.0))

print("Original False : {0} ({1:0.2f}%)".format(len(df.loc[df['diabetes'] == 0]), (len(df.loc[df['diabetes'] == 0])**/**len(df.index)) **\*** 100.0))

print("")

print("Training True : {0} ({1:0.2f}%)".format(len(y\_train[y\_train[:] == 1]), (len(y\_train[y\_train[:] == 1])**/**len(y\_train) **\*** 100.0)))

print("Training False : {0} ({1:0.2f}%)".format(len(y\_train[y\_train[:] == 0]), (len(y\_train[y\_train[:] == 0])**/**len(y\_train) **\*** 100.0)))

print("")

print("Test True : {0} ({1:0.2f}%)".format(len(y\_test[y\_test[:] == 1]), (len(y\_test[y\_test[:] == 1])**/**len(y\_test) **\*** 100.0)))

print("Test False : {0} ({1:0.2f}%)".format(len(y\_test[y\_test[:] == 0]), (len(y\_test[y\_test[:] == 0])**/**len(y\_test) **\*** 100.0)))

​

Original True : 268 (34.90%)

Original False : 500 (65.10%)

Training True : 188 (35.01%)

Training False : 349 (64.99%)

Test True : 80 (34.63%)

Test False : 151 (65.37%)

**Post-split Data Preparation**

Are these 0 values possible?

How many rows have have unexpected 0 values? How many rows have have unexpected 0 values.

df.head(10)

print("# rows in dataframe {0}".format(len(df)))

print("# rows missing glucose\_conc: {0}".format(len(df.loc[df['glucose\_conc'] == 0])))

print("# rows missing diastolic\_bp: {0}".format(len(df.loc[df['diastolic\_bp'] == 0])))

print("# rows missing thickness: {0}".format(len(df.loc[df['thickness'] == 0])))

print("# rows missing insulin: {0}".format(len(df.loc[df['insulin'] == 0])))

print("# rows missing bmi: {0}".format(len(df.loc[df['bmi'] == 0])))

print("# rows missing diab\_pred: {0}".format(len(df.loc[df['diab\_pred'] == 0])))

print("# rows missing age: {0}".format(len(df.loc[df['age'] == 0])))

# rows in dataframe 768

# rows missing glucose\_conc: 5

# rows missing diastolic\_bp: 35

# rows missing thickness: 227

# rows missing insulin: 374

# rows missing bmi: 11

# rows missing diab\_pred: 0

# rows missing age: 0

X\_train[2] *## note the first value has 0 value*

array([ 0. , 135. , 68. , 42. , 250. , 42.3 ,

0.365, 24. ])

**Impute with the mean**

**from** sklearn.preprocessing **import** Imputer

​

*#Impute with mean all 0 readings*

fill\_0 = Imputer(missing\_values=0, strategy="mean", axis=0)

​

X\_train = fill\_0.fit\_transform(X\_train)

X\_test = fill\_0.fit\_transform(X\_test)

X\_train[2] *# note the first value has mean value*

​

array([ 4.34056399, 135. , 68. , 42. ,

250. , 42.3 , 0.365 , 24. ])

**from** sklearn.naive\_bayes **import** GaussianNB

​

*# create Gaussian Naive Bayes model object and train it with the data*

nb\_model = GaussianNB()

​

nb\_model.fit(X\_train, y\_train.ravel())

GaussianNB(priors=None)

**Performance on Training Data**

*# predict values using the training data*

nb\_predict\_train = nb\_model.predict(X\_train)

​

*# import the performance metrics library*

**from** sklearn **import** metrics

​

*# Accuracy*

print("Accuracy: {0:.4f}".format(metrics.accuracy\_score(y\_train, nb\_predict\_train)))

Accuracy: 0.7542

**Performance on Testing Data**

*# predict values using the testing data*

nb\_predict\_test = nb\_model.predict(X\_test)

​

**from** sklearn **import** metrics

​

*# training metrics*

print("Accuracy: {0:.4f}".format(metrics.accuracy\_score(y\_test, nb\_predict\_test)))

Accuracy: 0.7359

**Metrics**

print("Print confusion matrix")

print("{0}".format(metrics.confusion\_matrix(y\_test, nb\_predict\_test)))

print("")

​

print("classfication report")

print(metrics.classification\_report(y\_test, nb\_predict\_test))

​

Print confusion matrix

[[118 33]

[ 28 52]]

classfication report

precision recall f1-score support

0 0.81 0.78 0.79 151

1 0.61 0.65 0.63 80

avg / total 0.74 0.74 0.74 231

**Confusion Matrix**

TN FP

FN TP

**Recall**

**Recall defines how well the model is predecting diabetes when the result is really diabetes.**

**recall = TP / (TP + FN)**

**precesion = TP / (TP + FP)**

**These two features should be greater than 70.**

**Random Forest Algorithm**

It is based on decision tree. It create multiple decision tree, hence forest name comes. It is an Ensamble Algorithm

**Train Random Forest Algorithm**

**from** sklearn.ensemble **import** RandomForestClassifier

rf\_model = RandomForestClassifier(random\_state=42) *# Create random forest object*

rf\_model.fit(X\_train, y\_train.ravel())

Out[125]:

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=10, n\_jobs=1, oob\_score=False, random\_state=42,

verbose=0, warm\_start=False)

**Performance on Training Data**

*# predict values using the training data*

rf\_predict\_train = rf\_model.predict(X\_train)

​

*# import the performance metrics library*

**from** sklearn **import** metrics

​

*# Accuracy*

print("Accuracy: {0:.4f}".format(metrics.accuracy\_score(y\_train, rf\_predict\_train)))

print()

Accuracy: 0.9870

**Performance on Testing Data**

rf\_predict\_test = rf\_model.predict(X\_test)

​

*# training metrics*

print("Accuracy: {0:.4f}".format(metrics.accuracy\_score(y\_test, rf\_predict\_test)))

Accuracy: 0.7100

**Metric**

print(metrics.confusion\_matrix(y\_test, rf\_predict\_test) )

print("")

print("Classification Report")

print(metrics.classification\_report(y\_test, rf\_predict\_test))

[[121 30]

[ 37 43]]

Classification Report

precision recall f1-score support

0 0.77 0.80 0.78 151

1 0.59 0.54 0.56 80

avg / total 0.70 0.71 0.71 231

**Notice that recall and precesion value is dropped to 0.56 and 0.59 respectively.**

This is due to the fact that the model is trained too well in traning data and this problem is called overfitting.

**Fixing Overfitting Problem.**

By manipulating Regualarization Hyperparameter of Algorithm we can choose upto what extent we want to precisely create decision boundary. Cross validation Both these approach can be used at the same time .

**Bias - variance trade - off**

This means we need to sacrifice some precision while creating decision boundary in training data so that our trained model works well will testing data set.

**Logistic Regression**

**from** sklearn.linear\_model **import** LogisticRegression

​

lr\_model =LogisticRegression(C=0.7, random\_state=42) *# hyper parameter value, the degree of extent we want to teach our model*

lr\_model.fit(X\_train, y\_train.ravel())

lr\_predict\_test = lr\_model.predict(X\_test)

​

*# training metrics*

print("Accuracy: {0:.4f}".format(metrics.accuracy\_score(y\_test, lr\_predict\_test)))

​

print(metrics.confusion\_matrix(y\_test, lr\_predict\_test) )

print("")

print("Classification Report")

​

print(metrics.classification\_report(y\_test, lr\_predict\_test))

Accuracy: 0.7446

[[128 23]

[ 36 44]]

Classification Report

precision recall f1-score support

0 0.78 0.85 0.81 151

1 0.66 0.55 0.60 80

avg / total 0.74 0.74 0.74 231

**Optimizing Regualarization parameter value.**

**Using the for loop to predict best possible value of C**

C\_start = 0.1

C\_end = 5

C\_inc = 0.1

​

C\_values, recall\_scores = [], []

​

C\_val = C\_start

best\_recall\_score = 0

**while** (C\_val **<** C\_end):

C\_values.append(C\_val)

lr\_model\_loop = LogisticRegression(C=C\_val, random\_state=42)

lr\_model\_loop.fit(X\_train, y\_train.ravel())

lr\_predict\_loop\_test = lr\_model\_loop.predict(X\_test)

recall\_score = metrics.recall\_score(y\_test, lr\_predict\_loop\_test)

recall\_scores.append(recall\_score)

**if** (recall\_score **>** best\_recall\_score):

best\_recall\_score = recall\_score

best\_lr\_predict\_test = lr\_predict\_loop\_test

C\_val = C\_val **+** C\_inc

​

best\_score\_C\_val = C\_values[recall\_scores.index(best\_recall\_score)]

print("1st max value of {0:.3f} occured at C={1:.3f}".format(best\_recall\_score, best\_score\_C\_val))

​

**%**matplotlib inline

plt.plot(C\_values, recall\_scores, "-")

plt.xlabel("C value")

plt.ylabel("recall score")

1st max value of 0.613 occured at C=1.400

Out[130]:

Text(0,0.5,'recall score')

**Remember we have more number of people who are non diabetes than diabetes. This imbalance is causing problem. There are hyper parameter which compensate these imbalances.**

**Logisitic regression with class\_weight="balanced"**

C\_start = 0.1

C\_end = 5

C\_inc = 0.1

​

C\_values, recall\_scores = [], []

​

C\_val = C\_start

best\_recall\_score = 0

**while** (C\_val **<** C\_end):

C\_values.append(C\_val)

lr\_model\_loop = LogisticRegression(C=C\_val, class\_weight="balanced", random\_state=42)

lr\_model\_loop.fit(X\_train, y\_train.ravel())

lr\_predict\_loop\_test = lr\_model\_loop.predict(X\_test)

recall\_score = metrics.recall\_score(y\_test, lr\_predict\_loop\_test)

recall\_scores.append(recall\_score)

**if** (recall\_score **>** best\_recall\_score):

best\_recall\_score = recall\_score

best\_lr\_predict\_test = lr\_predict\_loop\_test

C\_val = C\_val **+** C\_inc

​

best\_score\_C\_val = C\_values[recall\_scores.index(best\_recall\_score)]

print("1st max value of {0:.3f} occured at C={1:.3f}".format(best\_recall\_score, best\_score\_C\_val))

​

**%**matplotlib inline

plt.plot(C\_values, recall\_scores, "-")

plt.xlabel("C value")

plt.ylabel("recall score")

1st max value of 0.738 occured at C=0.300

Text(0,0.5,'recall score')

**from** sklearn.linear\_model **import** LogisticRegression

lr\_model =LogisticRegression( class\_weight="balanced", C=best\_score\_C\_val, random\_state=42)

lr\_model.fit(X\_train, y\_train.ravel())

lr\_predict\_test = lr\_model.predict(X\_test)

​

*# training metrics*

print("Accuracy: {0:.4f}".format(metrics.accuracy\_score(y\_test, lr\_predict\_test)))

print(metrics.confusion\_matrix(y\_test, lr\_predict\_test) )

print("")

print("Classification Report")

print(metrics.classification\_report(y\_test, lr\_predict\_test))

print(metrics.recall\_score(y\_test, lr\_predict\_test))

Accuracy: 0.7143

[[106 45]

[ 21 59]]

Classification Report

precision recall f1-score support

0 0.83 0.70 0.76 151

1 0.57 0.74 0.64 80

avg / total 0.74 0.71 0.72 231

0.7375

**LogisticRegressionCV**

**from** sklearn.linear\_model **import** LogisticRegressionCV

lr\_cv\_model = LogisticRegressionCV(n\_jobs=**-**1, random\_state=42, Cs=3, cv=10, refit=**False**, class\_weight="balanced") *# set number of jobs to -1 which uses all cores to parallelize*

lr\_cv\_model.fit(X\_train, y\_train.ravel())

Out[133]:

LogisticRegressionCV(Cs=3, class\_weight='balanced', cv=10, dual=False,

fit\_intercept=True, intercept\_scaling=1.0, max\_iter=100,

multi\_class='ovr', n\_jobs=-1, penalty='l2', random\_state=42,

refit=False, scoring=None, solver='lbfgs', tol=0.0001,

verbose=0)

**Predict on Test data**

lr\_cv\_predict\_test = lr\_cv\_model.predict(X\_test)

​

*# training metrics*

print("Accuracy: {0:.4f}".format(metrics.accuracy\_score(y\_test, lr\_cv\_predict\_test)))

print(metrics.confusion\_matrix(y\_test, lr\_cv\_predict\_test, labels=[1, 0]) )

print("")

print("Classification Report")

print(metrics.classification\_report(y\_test, lr\_cv\_predict\_test, labels=[1,0]))

Accuracy: 0.7013

[[ 54 26]

[ 43 108]]

Classification Report

precision recall f1-score support

1 0.56 0.68 0.61 80

0 0.81 0.72 0.76 151

avg / total 0.72 0.70 0.71 231