Machine Learning Assignment

Identification of diabetes patients using Naive Bayes Algorithm, Random Forest Classifier Algorithm, Logistic regression & Cross validation.

SE4060
Bachelor of Science (Honors) in Information Technology
Sri Lanka Institute of Information Technology
Sri Lanka.

IT16155794
R.M.S.M.Rathnayake

Content……

1. Introduction

- For many years, humans usually survived as a food shortage for carbohydrates. In the past few years, due to the sudden drop in physical activity due to a sudden transfer to traditional agricultural products, they have been involved in most studies due to the high prevalence of Type 2 diabetes.
- We use **Naive Bayes Algorithm, Random Forest Classifier Algorithm, Logistic regression & Cross validation to do this prediction.**

- **We use Python language and Jupyter notebook for our implementation.**
- **Python Libraries**
    - ❏ **Numpy - scientific computing**
    - ❏ **Pandas - data frames**
    - ❏ **Matplotlib - 2D plotting**
    - ❏ **Scikit-learn - Algorithms, Pre-processing, Performance evaluation**

- **Jupyter Notebook**
    - ❏ **Formerly Ipython Notebook**
    - ❏ **Notebooks contain code and text**
    - ❏ **Perfect for iterable work like Machine Learning**
    - ❏ **Shareable**
    - ❏ **Support multiple languages**

Figure 1 - Machine Learning Workflow

2. Data Preparation

2.1 Introduction to data

- **In here we use Pima Indian data to create a prediction model. This model must predict which people are likely to develop diabetes with 70% or greater accuracy.**

- **Activites**
  - ❑ **Find the data we need**
  - ❑ **Inspect and clean the data**
  - ❑ **Explore the data**
  - ❑ **Model the data to Tidy data**

- **Pima Indian diabetes data**
  - ❑ **pima -data.csv - based on UCI data**
  - ❑ **Female patients at least 21 years old**
  - ❑ **768 patient observation rows**
  - ❑ **10 columns - 9 feature columns & 1 class column**

2.2 Load and review data

- **We can find our dataset in here.**
  - ❑ https://www.kaggle.com/uciml/pima-indians-diabetes-database

- **Load and review data**

Load and review data

```
In [5]: df = pd.read_csv("./data/pima-data.csv")      # load Pima data.  Adjust path as necessary

In [ ]:
```

- **Shape, Head and Tail**

```
In [6]: df.shape
Out[6]: (768, 10)

In [7]: df.head(5)
Out[7]:
```

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	diabetes
0	6	148	72	35	0	33.6	0.627	50	1.3790	True
1	1	85	66	29	0	26.6	0.351	31	1.1426	False
2	8	183	64	0	0	23.3	0.672	32	0.0000	True
3	1	89	66	23	94	28.1	0.167	21	0.9062	False
4	0	137	40	35	168	43.1	2.288	33	1.3790	True

```
In [8]: df.tail(5)
```

Out[8]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	diabetes
763	10	101	76	48	180	32.9	0.171	63	1.8912	False
764	2	122	70	27	0	36.8	0.340	27	1.0638	False
765	5	121	72	23	112	26.2	0.245	30	0.9062	False
766	1	126	60	0	0	30.1	0.349	47	0.0000	True
767	1	93	70	31	0	30.4	0.315	23	1.2214	False

2.3 Columns to Eliminate

1. **Not used**
2. **No values**
3. **Duplicates**

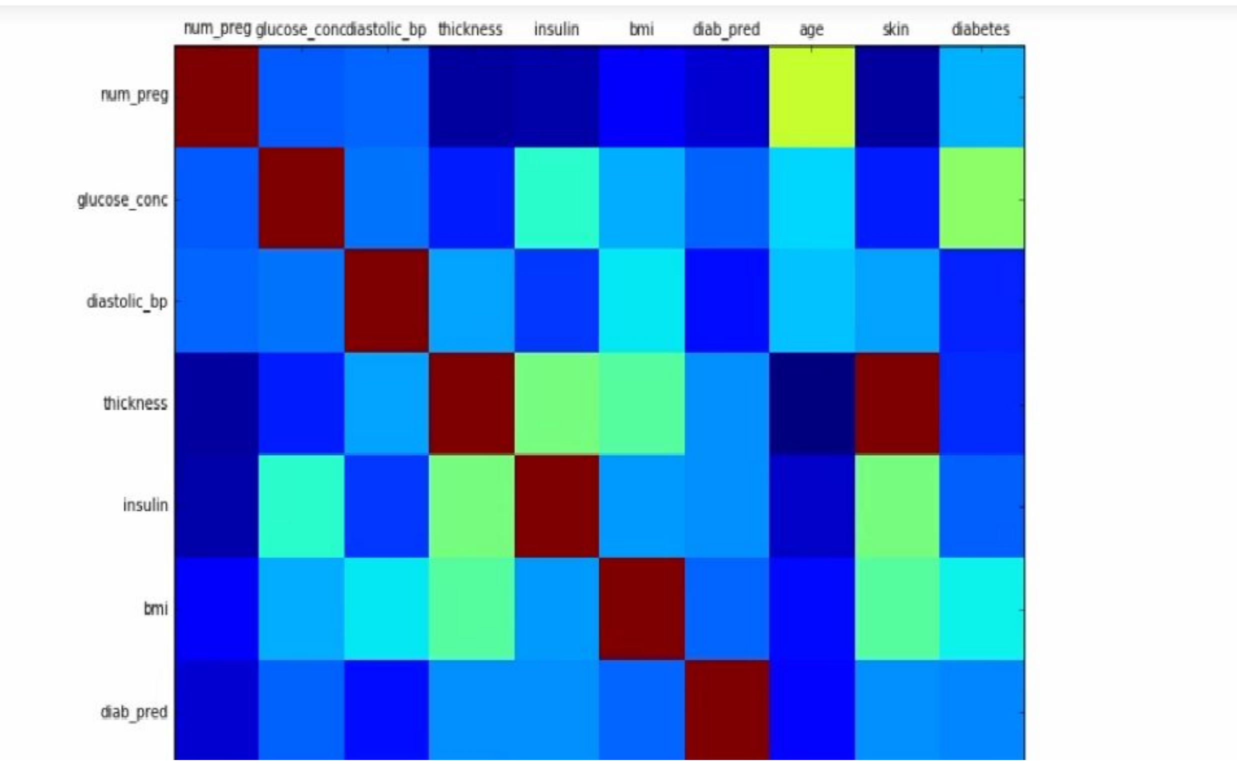
- **Check for null values**

Check for null values

```
In [19]: df.isnull().values.any()

Out[19]: False
```

- **Check for correlated values**
    - ❑ **We use Matplotlib library in here.**

In [9]:
```python
def plot_corr(df, size=11):
    """
    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.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
```

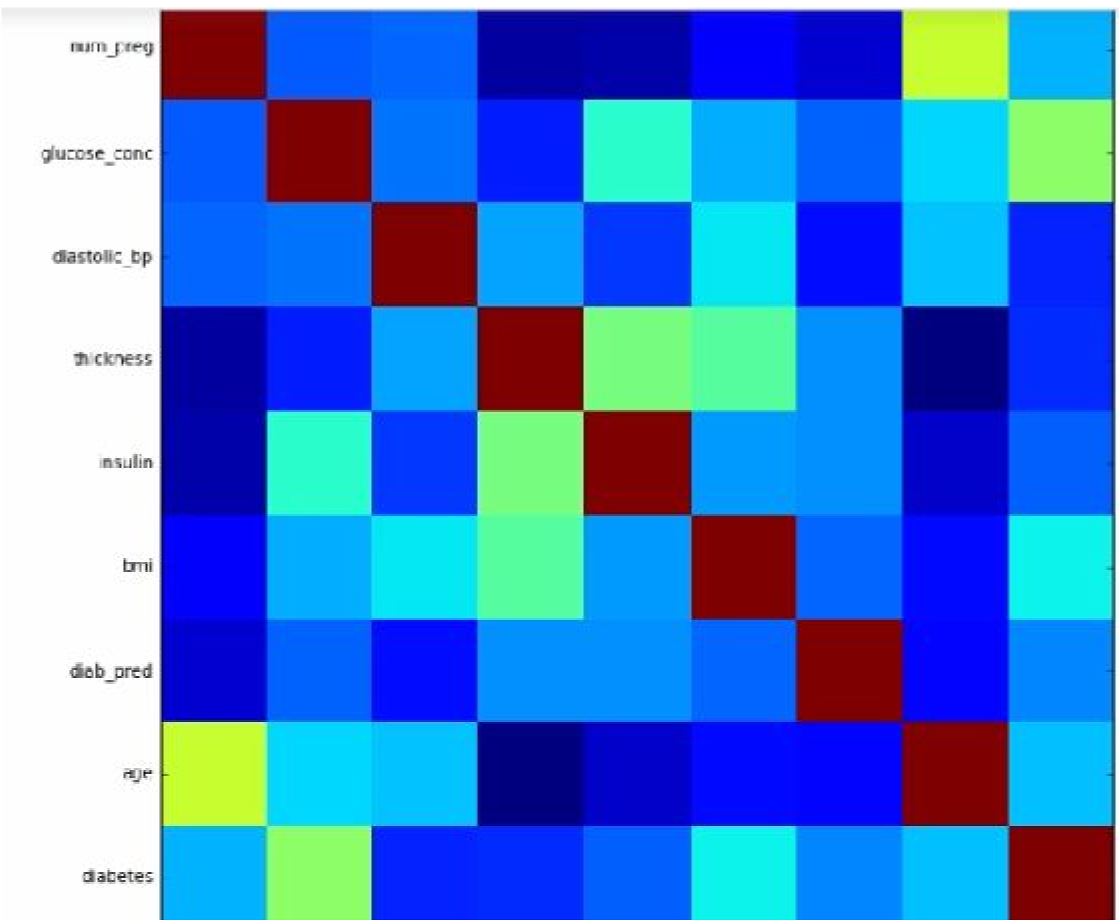


```
In [18]: del df['skin']
```

```
In [19]: df.head()
```

Out[19]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	True
1	1	85	66	29	0	26.6	0.351	31	False
2	8	183	64	0	0	23.3	0.672	32	True
3	1	89	66	23	94	28.1	0.167	21	False
4	0	137	40	35	168	43.1	2.288	33	True



3. Molding the data

- **In here we are trying to adjusting our data types.**

3.1 Check the data types

Check Data Types

In [21]:
```
df.head(5)
```

Out[21]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	True
1	1	85	66	29	0	26.6	0.351	31	False
2	8	183	64	0	0	23.3	0.672	32	True
3	1	89	66	23	94	28.1	0.167	21	False
4	0	137	40	35	168	43.1	2.288	33	True

3.2 Change True to 1 and False to 0

Change True to 1, False to 0

In [26]:
```
diabetes_map = {True : 1, False : 0}
```

In [27]:
```
df['diabetes'] = df['diabetes'].map(diabetes_map)
```

In [28]:
```
df.head(5)
```

Out[28]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

- **We use map method in pandas framework to do this.**

3.3 Check True False ratio

Check true/false ratio

In [31]:
```
num_true = len(df.loc[df['diabetes'] == True])
num_false = len(df.loc[df['diabetes'] == False])
print("Number of True cases:  {0} ({1:2.2f}%)".format(num_true, (num_true/ (num_true + num_false)) * 100))
print("Number of False cases: {0} ({1:2.2f}%)".format(num_false, (num_false/ (num_true + num_false)) * 100))
```

```
Number of True cases:  268 (34.90%)
Number of False cases: 500 (65.10%)
```

4. Selecting the algorithm

- **There are four type of Algorithm decision factors as Learning type, result, complexity and basic vs enhanced. There are 50 algorithms at the beginning.**

- **In here we use this data to create a prediction model and develop diabetes with 70% or greater accuracy. So we use Supervised machine learning to do this. Now we have only 28 algorithms.**

- **When we consider about the result type we have two types as Regression and Classification.We are predicting a binary outcome, diabetes or not so we use Classification in here. Now we have only 20 algorithms.**

- **We use Boost performance in complexity section because of simplicity. Now we have only 14 algorithms.**

- **Three candidate algorithms after this filtering.**
  1. **Naive Bayes**
  2. **Logistic Regression**
  3. **Decision Tree**

- **Why we use the Naive Bayes ?**
  1. **Simple - easy to understand**
  2. **Fast - up to 100X faster**
  3. **Stable to data changes**

5. Training

5.1 Introduction to training

- **Letting specific data teach a Machine Learning algorithm to create a specific prediction model.**

- **There are two reasons for retrain**
    1. **New data => better prediction**
    2. **Verify training performance with new data**

5.2 Introduction to Scikit - learn library

- **Designed to work with Numpy, SciPy and Pandas.**

- **Common interface across algorithms.**

- **Toolset for training and evaluation tasks**
    1. **Data splitting**
    2. **Pre - processing**
    3. **Feature selection**
    4. **Model training**
    5. **Model tuning**

5.3 Splitting the data

5.3.1 Splitting the data

Spliting the data

70% for training, 30% for testing

In [19]:
```python
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
```

5.3.2 Check to ensure

We check to ensure we have the the desired 70% train, 30% test split of the data

In [20]:
```python
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
```

5.3.3 Verify for the predicted values

Verifying predicted value was split correctly

In [21]:
```python
print("Original True  : {0} ({1:0.2f}%)".format(len(df.loc[df['diabetes'] == 1]), (len(df.loc[df['diabetes'] == 1])/len(df.ind
print("Original False : {0} ({1:0.2f}%)".format(len(df.loc[df['diabetes'] == 0]), (len(df.loc[df['diabetes'] == 0])/len(df.ind
print("")
print("Training True  : {0} ({1:0.2f}%)".format(len(y_train[y_train[:] == 1]), (len(y_train[y_train[:] == 1])/len(y_train) * 1
print("Training False : {0} ({1:0.2f}%)".format(len(y_train[y_train[:] == 0]), (len(y_train[y_train[:] == 0])/len(y_train) * 1
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%)
```

5.4 Missing values

- **Find the missing values**

Are these 0 values possible?

How many rows have have unexpected 0 values?

```
In [24]: 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
```

- **Options for missing values**

    1. **Ignore**
    2. **Drop observation (rows)**
    3. **Replace values (Impute)**

- **We have 768 rows and 374 missing insulin values. So we can't ignore/delete 50% of data.**

- **Imputing is the best solution for missing values**

    - **Replace with mean, median**
    - **Replace with expert knowledge derives value**

- **Impute with the mean**

Impute with the mean

In [25]:
```
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)
```

5.4 Training the algorithm

- **Training initial algorithm - Naive Bayes**

Training Initial Algorithm - Naive Bayes

In [26]:
```
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())
```

Out[26]:
```
GaussianNB()
```

6. Testing

6.1 Performance of training data

Performance on Training Data

```
In [26]: # 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)))
print()

Accuracy: 0.7542
```

6.2 Performance of testing data

Performance on Testing Data

```
In [27]: # 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
```

6.3 Metrics

Metrics

In [29]:
```python
print("Confusion Matrix")
print("{0}".format(metrics.confusion_matrix(y_test, nb_predict_test)))
print("")

print("Classification Report")
print(metrics.classification_report(y_test, nb_predict_test))
```

```
Confusion Matrix
[[118  33]
 [ 28  52]]

Classification 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**
  - **118 - True Negative values (TN)**
  - **33 - False Positive values (FP)**
  - **28 - False Negative values (FN)**
  - **52 - True Positive values (TP)**

- **Precision = TP / (TP + FP)**

- **Recall = TP / (TP + FN)**

6.4 Performance Improvement Options

- **Adjust current algorithm**

- **Get more data or improve data**

- **Improve training**
- **Switch algorithms**

6.5 Random Forest Algorithm

6.5.1 Why we use this?

- **Ensemble Algorithm**
- **Fits multiple trees with subsets of data**
- **Averages tree results to improve performance and control overfitting**

6.5.2 Use the algorithm

Random Forest

```
In [30]: from sklearn.ensemble import RandomForestClassifier
         rf_model = RandomForestClassifier(random_state=42)      # Create random forest object
         rf_model.fit(X_train, y_train.ravel())

Out[30]: 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)
```

6.5.3 Predict training and testing data

Predict Training Data

```
In [31]: rf_predict_train = rf_model.predict(X_train)
         # training metrics
         print("Accuracy: {0:.4f}".format(metrics.accuracy_score(y_train, rf_predict_train)))

Accuracy: 0.9870
```

Predict Test Data

```
In [32]: 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
```

6.5.4 Metrics

```
In [33]: 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
```

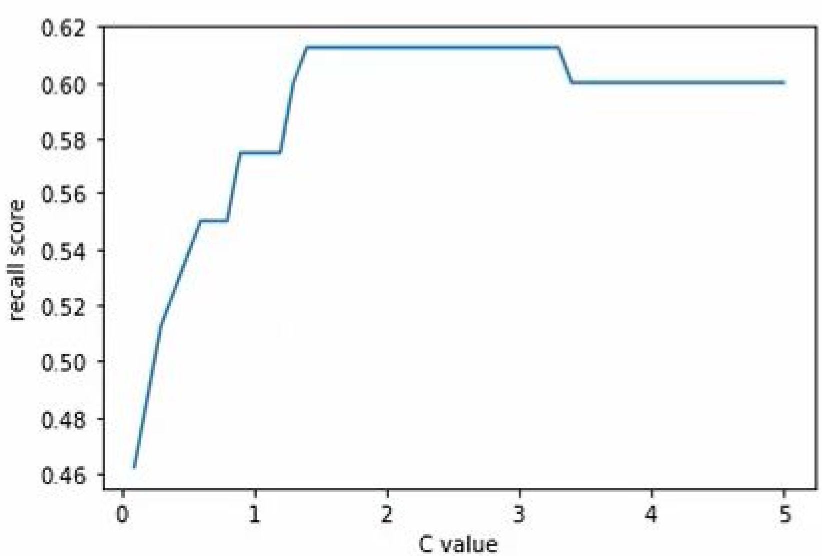
6.6 Logistic Regression

Logistic Regression

In [35]:
```
from sklearn.linear_model import LogisticRegression

lr_model =LogisticRegression(C=0.7, 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))
```

```
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
```

6.6.1 Setting Regularization parameter (changing ‘c’ values)



6.6.2 Fixing unbalanced classes

- **Change class weight = “balanced” and C = best_score_C_val**

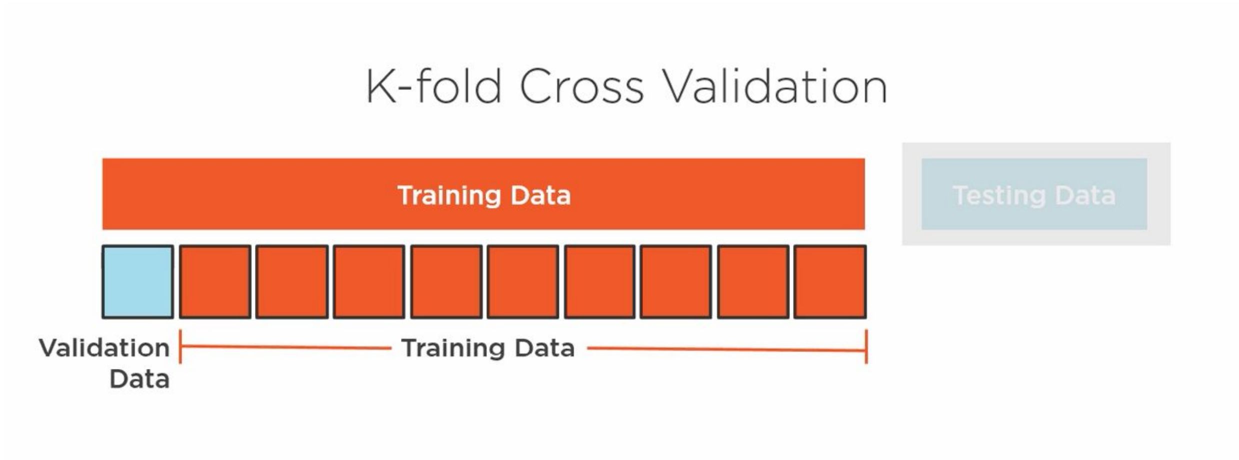
```
In [ ]: from sklearn.linear_model import LogisticRegression
        lr_model =LogisticRegression( class_weight="balanced", C=best_score_C_val, random_state=42)
        lr_model.fit(X_train, y_train.ravel())
        lr_predict_test = lr_model.predict(X_test)

        # 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))
```

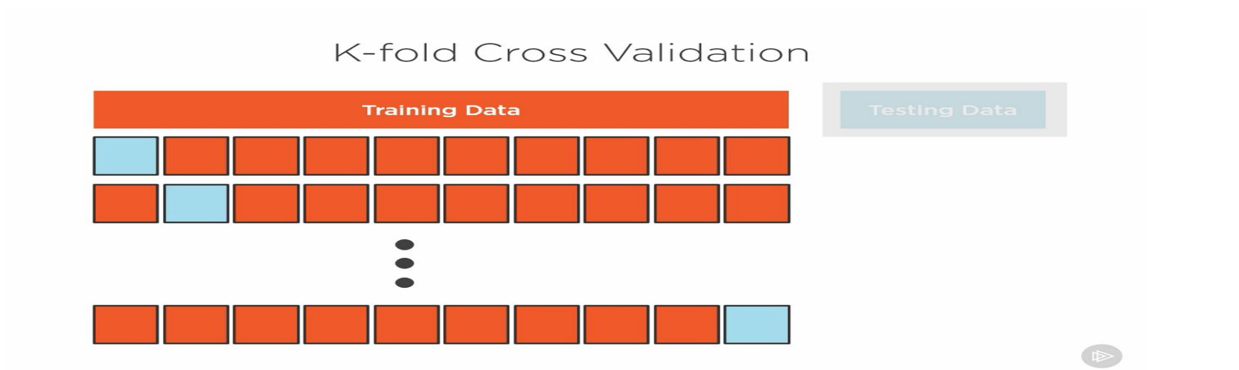
- **Final result of Metrics after unbalanced classes**

```
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
```

6.7 Cross Validation

6.7.1 k - fold cross validation





6.7.2 See availability of the logistic regression parameters

LogisticRegressionCV

```
In [39]: 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 numbe
         lr_cv_model.fit(X_train, y_train.ravel())

Out[39]: 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)
```

6.7.3 Predict on Test data

Predict on Test data

```
In [40]: 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) )
         print("")
         print("Classification Report")
         print(metrics.classification_report(y_test, lr_cv_predict_test))
```

```
Accuracy: 0.7143
[[110  41]
 [ 25  55]]

Classification Report
             precision    recall  f1-score   support

          0       0.81      0.73      0.77       151
          1       0.57      0.69      0.62        80

avg / total       0.73      0.71      0.72       231
```

7. Using your trained Model

- **The truncated file contained 4 rows from the original CSV.**
- **Data is the same is in same format as the original CSV file's data. Therefore, just like the original data, we need to transform it before we can make predictions on the data.**
- **Note: If the data had been previously "cleaned up" this would not be necessary.**
- **We do this by executed the same transformations as we did to the original data**
- **Start by dropping the "skin" which is the same as thickness, with different units.**
- **Data has 0 in places it should not. Just like test or test datasets we will use imputation to fix this.**

8. Appendix

8.1 Code

```
##Import some basic libraries.

import pandas as pd               # pandas is a dataframe library
import matplotlib.pyplot as plt    # matplotlib.pyplot plots data

%matplotlib inline


##Loading and Reviewing the Data

df = pd.read_csv("./data/pima-data.csv")

df.shape

df.head(5

df.tail(5)


##Check for null values

df.isnull().values.any()


##Correlated Feature Check

def plot_corr(df, size=11):
    """
    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.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.corr()

df.head(5)

del df['skin']

df.head(5)

plot_corr(df)


##Change diabetes from boolean to integer, True=1, False=0

diabetes_map = {True : 1, False : 0}
df['diabetes'] = df['diabetes'].map(diabetes_map)


##Check for null values

df.isnull().values.any()


##Check class distribution

num_obs = len(df)
num_true = len(df.loc[df['diabetes'] == 1])
num_false = len(df.loc[df['diabetes'] == 0])
print("Number of True cases:  {0} ({1:2.2f}%)".format(num_true, (num_true/num_obs) * 100))
print("Number of False cases: {0} ({1:2.2f}%)".format(num_false, (num_false/num_obs) * 100))


##Spliting the data
###70% for training, 30% for testing

#from sklearn.cross_validation import train_test_split
from sklearn.model_selection 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

###We check to ensure we have the the desired 70% train, 30% test split of the data

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))
```

##Verifying predicted value was split correctly

```
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)))
```

##Post-split Data Preparation

###Are these 0 values possible?

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

```
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])))
```

##Impute with the mean

```
# NEED CALLOUT MENTION CHANGE TO SIMPLEIMPUTER
#from sklearn.preprocessing import Imputer
from sklearn.impute import SimpleImputer
#Impute with mean all 0 readings
#fill_0 = Imputer(missing_values=0, strategy="mean", axis=0)
fill_0 = SimpleImputer(missing_values=0, strategy="mean")
X_train = fill_0.fit_transform(X_train)
X_test = fill_0.fit_transform(X_test)
```

##Training Initial Algorithm - Naive Bayes

```
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())


##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)))
print()


##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("nb_predict_test", nb_predict_test)
print ("y_test", y_test)
print("Accuracy: {0:.4f}".format(metrics.accuracy_score(y_test, nb_predict_test)))


##Metrics

print("Confusion Matrix")
print("{0}".format(metrics.confusion_matrix(y_test, nb_predict_test)))
print("")

print("Classification Report")
print(metrics.classification_report(y_test, nb_predict_test))


##Random Forest

from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(random_state=42, n_estimators=10)         # Create random forest
object
rf_model.fit(X_train, y_train.ravel())


##Predict Training Data

rf_predict_train = rf_model.predict(X_train)
# training metrics
print("Accuracy: {0:.4f}".format(metrics.accuracy_score(y_train, rf_predict_train)))
```

```
##Predict Test Data

rf_predict_test = rf_model.predict(X_test)

### training metrics
print("Accuracy: {0:.4f}".format(metrics.accuracy_score(y_test, rf_predict_test))

print(metrics.confusion_matrix(y_test, rf_predict_test) )
print("")
print("Classification Report")
print(metrics.classification_report(y_test, rf_predict_test))


##Logistic Regression

from sklearn.linear_model import LogisticRegression

lr_model =LogisticRegression(C=0.7, random_state=42, solver='liblinear', max_iter=10000)
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))


###Setting regularization parameter


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, solver='liblinear')
    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")


##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, solver='liblinear', max_iter=10000)
    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")

from sklearn.linear_model import LogisticRegression
lr_model =LogisticRegression( class_weight="balanced", C=best_score_C_val, random_state=42, solver='liblinear')
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))
```

```
##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", max_iter=500)  # set number of jobs to -1 which uses all cores to parallelize
lr_cv_model.fit(X_train, y_train.ravel())


##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) )
print("")
print("Classification Report")
print(metrics.classification_report(y_test, lr_cv_predict_test))


##Using your trained Model

###Save trained model to file

from sklearn.externals import joblib
joblib.dump(lr_cv_model, "./data/pima-trained-model.pkl")


#Load trained model from file

lr_cv_model = joblib.load("./data/pima-trained-model.pkl")


###Test Prediction on data

# get data from truncated pima data file
df_predict = pd.read_csv("./data/pima-data-trunc.csv")
print(df_predict.shape)

df_predict
del df_predict['skin']
df_predict

X_predict = df_predict
del X_predict['diabetes']

#Impute with mean all 0 readings
from sklearn.impute import SimpleImputer
fill_0 = SimpleImputer(missing_values=0, strategy="mean") #, axis=0)
X_predict = fill_0.fit_transform(X_predict)
```

```
##Predict diabetes with the prediction data. Returns 1 if True, 0 if false

lr_cv_model.predict(X_predict)
```