CS564 Foundations of Machine Learning
Assignment: 1 Part: B

Mukuntha N S (1601CS27)
R J Srivatsa (1601CS35)
Shikhar Jaiswal (1601CS44)

August 24, 2019

1 Assignment Description

We are given a dataset (**data.csv**), which has 8 variables as **NumPreg, PlasmaGlucose, DiastolicBP, TricepSkin, BodyMassIndex, Pedigree, Age, Diabetic**. The target is to fix a logistic regression model to predict the **Diabetic** variable based on the other 7 variables.

2 Procedure

Installation

Install the following dependencies either using pip or through conda in a Python 3.5+ environment:
- jupyter
- pandas
- numpy
- sklearn
- seaborn
- matplotlib

```
python3 -m pip install jupyter pandas numpy sklearn seaborn matplotlib
```

or alternatively,

```
conda install -c anaconda jupyter pandas numpy sklearn seaborn matplotlib
```

Running The Notebook

To run the program, head to the directory $Assignment1/Q2$. Please note that the **dataset should be present in the same directory** as the jupyter notebook. Use the following command to run the notebook:

```
jupyter notebook Q2.ipynb
```

3 Discussion

The primary objective of the assignment is to develop the best model and the optimal threshold to predict the categorical response variable **Diabetic** in case of the given dataset. The following sub-sections contain the explanation for individual code snippets. For a detailed look at the output, please refer the notebook and the individual files provided.

Notebook Code

Pre-processing and Visualization

Import the Python dependencies, set the random seed and check the data samples and their values.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
from itertools import combinations
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import KFold
from sklearn.decomposition import PCA

random.seed(21)
np.random.seed(21)

data = pd.read_csv('./data.csv', encoding = 'utf-8')

print(data)
```

Exploratory Data Analysis

We convert the string based categorical labels to integer based values, and then normalize the remaining variable columns. We also generate the heatmap of correlation values of different column variables.

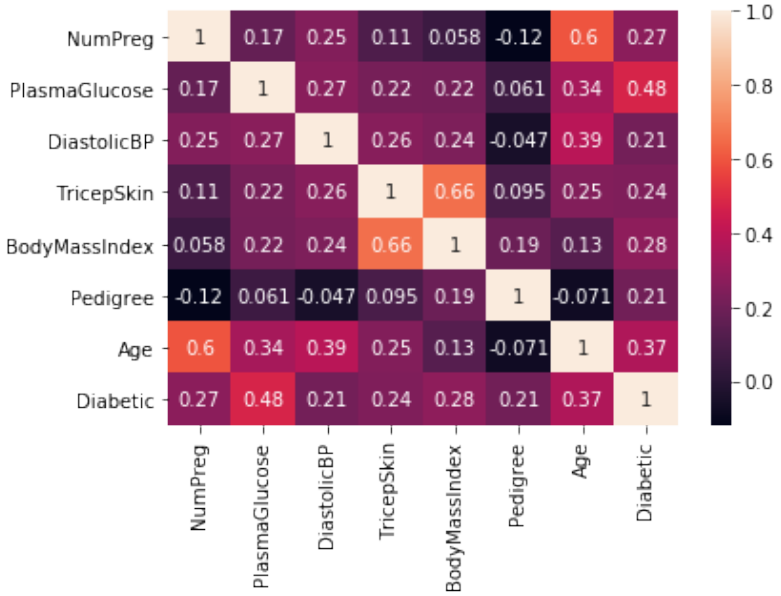
```
metrics = ['NumPreg', 'PlasmaGlucose', 'DiastolicBP', 'TricepSkin', 'BodyMassIndex',
           'Pedigree', 'Age']

data['Diabetic'] = data['Diabetic'].apply(lambda x: 0 if x == 'No' else 1)

data[metrics] = data[metrics].apply(lambda column: (column - column.mean()) / column.std(),
                                    axis = 0)

correlation_heatmap = data.corr()
sns.heatmap(correlation_heatmap, annot = True, xticklabels = correlation_heatmap.columns,
            yticklabels = correlation_heatmap.columns)
```

Plot



Helper Functions

Function for generating all combinations of features from a given feature list.

```
def sublists(input_list):
    subs = []

    for i in range(0, len(input_list) + 1):
        temp = [list(x) for x in combinations(input_list, i)]

        if len(temp) > 0:
            subs.extend(temp)

    return subs
```

Function for generating the graphs for the most dominant features affecting a given model's output.

```
def grapher(coefficient, labels):
    features = pd.DataFrame()
    features['Features'] = labels
    features['Importance'] = coefficient
    features.sort_values(by = ['Importance'], ascending = True, inplace = True)
    features['Positive'] = features['Importance'] > 0
    features.set_index('Features', inplace = True)
    features.Importance.plot(kind = 'barh', figsize = (11, 6),
                             color = features.Positive.map({True: 'blue', False: 'red'}))
    plt.xlabel('Importance')
```

Function for fitting the data to a Logistic Regression model without thresholding, and checking the average accuracy value.

```
def tester(X, Y, iterations = 20, graph = False):
    kf = KFold(n_splits = 10, shuffle = True)
    scores = []
    mean_scores = []

    for i in range(iterations):
        for train_index, test_index in kf.split(X):
            train_length = len(train_index)
            valid_index = train_index[: train_length // 10]
            train_index = train_index[train_length // 10 :]
            X_train, X_test = X.iloc[train_index].drop(['index'], axis = 1),
                              X.iloc[test_index].drop(['index'], axis = 1)
            Y_train, Y_test = Y.iloc[train_index].drop(['index'], axis = 1),
                              Y.iloc[test_index].drop(['index'], axis = 1)
            clf = linear_model.LogisticRegression(solver = 'liblinear', penalty = 'l2',
                  max_iter = 200).fit(X_train, Y_train.values.ravel())

            if graph:
                grapher(list(clf.coef_[0]), list(X_train.columns))
                graph = False

            scores.append(clf.score(X_test, Y_test))

        mean_scores.append(np.mean(scores))

    return np.mean(mean_scores)
```

Function for fitting the data to a Logistic Regression model with thresholding, and checking the average accuracy and threshold values.

```
def tester_with_threshold(X, Y, iterations = 20, graph = False):
    kf = KFold(n_splits = 10, shuffle = True)
    scores = []
    mean_scores = []
    best_thresholds = []
    mean_thresholds = []
    threshold_choices = np.arange(0.1, 0.8, 0.05)

    for i in range(iterations):
        for train_index, test_index in kf.split(X):
            train_length = len(train_index)
            valid_index = train_index[: train_length // 10]
            train_index = train_index[train_length // 10 :]

            if isinstance(X, np.ndarray):
```

```
            X_train, X_valid, X_test = X[train_index], X[valid_index], X[test_index]
            Y_train, Y_valid, Y_test = Y[train_index], Y[valid_index], Y[test_index]
            clf = linear_model.LogisticRegression(solver = 'lbfgs', penalty = 'l2',
                  max_iter = 200).fit(X_train, Y_train.ravel())
        elif isinstance(X, pd.DataFrame):
            X_train, X_valid, X_test = X.iloc[train_index].drop(['index'], axis = 1),
                                       X.iloc[valid_index].drop(['index'], axis = 1),
                                       X.iloc[test_index].drop(['index'], axis = 1)
            Y_train, Y_valid, Y_test = Y.iloc[train_index].drop(['index'], axis = 1),
                                       Y.iloc[valid_index].drop(['index'], axis = 1),
                                       Y.iloc[test_index].drop(['index'], axis = 1)
            clf = linear_model.LogisticRegression(solver = 'lbfgs', penalty = 'l2',
                  max_iter = 200).fit(X_train, Y_train.values.ravel())
            Y_valid = Y_valid.values
            Y_test = Y_test.values
        else:
            raise TypeError("X")

        if graph:
            grapher(list(clf.coef_[0]), list(X_train.columns))
            graph = False

        valid_probability = clf.predict_proba(X_valid)[:, 1]
        best_threshold = 0.0
        best_accuracy = 0.0

        for threshold in threshold_choices:
            accuracy = accuracy_score(Y_valid.ravel() == 1,
                                      valid_probability >= threshold)

            if accuracy > best_accuracy:
                best_accuracy = accuracy
                best_threshold = threshold

        best_thresholds.append(best_threshold)
        scores.append(accuracy_score(Y_test.ravel() == 1,
                      clf.predict_proba(X_test)[:, 1] >= best_threshold))

    mean_scores.append(np.mean(scores))
    mean_thresholds.append(np.mean(best_thresholds))

return np.mean(mean_scores), np.mean(mean_thresholds)
```

Simple Analysis

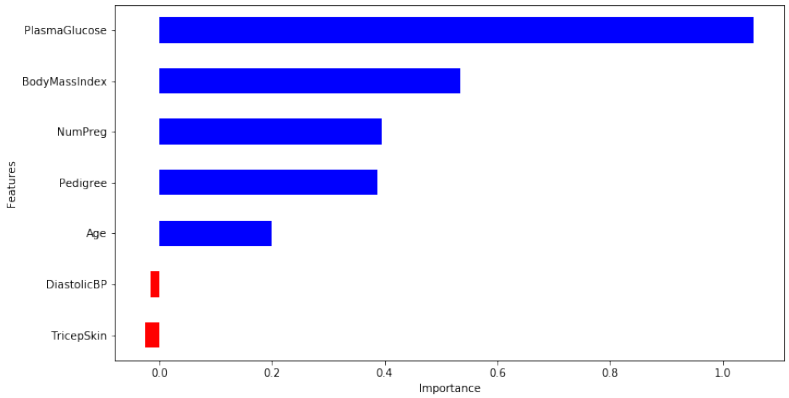
We fit all the variables to the simple Logistic Regression model, and check for accuracy and impactfulness of different features.

```
X = data[metrics].reset_index()
Y = data['Diabetic'].reset_index()
score = tester(X, Y, 1, True)

print(metrics)
print(score)
```

```
Output:
['NumPreg', 'PlasmaGlucose', 'DiastolicBP', 'TricepSkin', 'BodyMassIndex', 'Pedigree',
 'Age']
0.745
```

Plot



Logistic Regression

We fit all combinations of the features to the simple Logistic Regression model, and check for accuracy of different feature sets. We finally chose the feature set with the maximum average accuracy 10 fold cross-validation. We find that given all possible subset of features, our model performs best on three specific features, with a 10 fold cross-validation accuracy of about 78%.

```
power_metrics = sublists(metrics)
max_score = -1

for metric_list in power_metrics:
    if metric_list == []:
        continue

    X = data[metric_list].reset_index()
    Y = data['Diabetic'].reset_index()
    score = tester(X, Y, 1, False)

    if score > max_score:
```

```
        max_score = score
        print(metric_list)
        print(max_score)
```

```
Output:
['NumPreg']
0.6850000000000002
['PlasmaGlucose']
0.755
['PlasmaGlucose', 'DiastolicBP', 'Pedigree']
0.77
['PlasmaGlucose', 'BodyMassIndex', 'Pedigree']
0.78
```

Logistic Regression with Thresholding

We fit all combinations of the features to the Logistic Regression with Thresholding model, and check for accuracy of different feature sets. We finally chose the feature set with the maximum average accuracy 10 fold cross-validation. We base our choice of optimal threshold on the performance achieved on the validation set data. We find that given all possible subset of features, our model performs best on six specific features, with a 10 fold cross-validation accuracy of about 78% with a threshold value of 0.48.

```
max_score = -1

for metric_list in power_metrics:
    if metric_list == []:
        continue

    X = data[metric_list].reset_index()
    Y = data['Diabetic'].reset_index()
    score, threshold = tester_with_threshold(X, Y, 1, False)

    if score > max_score:
        max_score = score
        print(metric_list)
        print(max_score, threshold)
```

```
Output:
['NumPreg', 'PlasmaGlucose', 'DiastolicBP']
0.6100000000000001 0.21000000000000005
['NumPreg', 'PlasmaGlucose', 'TricepSkin']
0.7 0.31000000000000005
['NumPreg', 'PlasmaGlucose', 'Pedigree']
0.7150000000000001 0.33500000000000013
['NumPreg', 'BodyMassIndex', 'Pedigree']
0.72 0.5500000000000003
['NumPreg', 'Pedigree', 'Age']
```

```
0.7550000000000001 0.4750000000000001
['NumPreg', 'PlasmaGlucose', 'TricepSkin', 'Pedigree', 'Age']
0.765 0.43000000000000016
['NumPreg', 'PlasmaGlucose', 'DiastolicBP', 'TricepSkin', 'Pedigree', 'Age']
0.78 0.48000000000000015
```

Principal Components Analysis

We transform all features of the dataset using PCA and score the different size of subsets. We finally chose the subset size with the maximum average accuracy over 10 fold cross-validation. We find that given all possible number of features, our model performs best on three major features, with a 10 fold cross-validation accuracy of about 78% as well.

```
X = data[metrics]
Y = data['Diabetic'].reset_index()

pca = PCA(n_components = 7, whiten = True)
X = pd.DataFrame(pca.fit_transform(X), index = None).reset_index()

for num_components in range(1, 8):
    X_components = pd.DataFrame(X.values[:, 1 : num_components + 1],
                                index = None).reset_index()
    score = tester(X_components, Y, 1, False)
    print('No. of components: {}, Score: {}'.format(num_components, score))
```

```
Output:
No. of components: 1, Score: 0.73
No. of components: 2, Score: 0.725
No. of components: 3, Score: 0.78
No. of components: 4, Score: 0.7699999999999999
No. of components: 5, Score: 0.76
No. of components: 6, Score: 0.7499999999999999
No. of components: 7, Score: 0.7650000000000001
```

Thus we observe that different techniques in the model converge to the same neighbourhood of accuracy values, given the Logistic Regression algorithm. Hence we can conclude, that this is the optimal performance that is achievable empirically.