Diabetes dataset

Predict if a person is at risk of developing diabetes

Binary Classification problem - XGBoost

In [1]:
```
# Install xgboost in notebook instance.
#### Command to install xgboost
!pip install xgboost
```

```
Looking in indexes: https://pypi.org/simple, https://pip.repos.neuron.amazonaws.com
Collecting xgboost
  Downloading xgboost-1.7.6-py3-none-manylinux2014_x86_64.whl (200.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 200.3/200.3 MB 3.1 MB/s eta 0:00:0000:0100:01
Requirement already satisfied: numpy in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from xgb
oost) (1.22.3)
Requirement already satisfied: scipy in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from xgb
oost) (1.10.1)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.6
```

In [2]:
```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import itertools

import xgboost as xgb
from sklearn.metrics import classification_report, confusion_matrix
```

In [3]:
```
column_list_file = 'diabetes_train_column_list.txt'
train_file = 'diabetes_train.csv'
validation_file = 'diabetes_validation.csv'
```

In [4]:
```
columns = ''
with open(column_list_file,'r') as f:
    columns = f.read().split(',')
```

In [5]:
```
columns
```

Out[5]:
```
['diabetes_class',
 'preg_count',
 'glucose_concentration',
 'diastolic_bp',
 'triceps_skin_fold_thickness',
 'two_hr_serum_insulin',
 'bmi',
 'diabetes_pedi',
 'age']
```

In [6]:
```
# Specify the column names as the file does not have column header
df_train = pd.read_csv(train_file,names=columns)
df_validation = pd.read_csv(validation_file,names=columns)
```

In [7]:
```
df_train.head()
```

Out[7]:

	diabetes_class	preg_count	glucose_concentration	diastolic_bp	triceps_skin_fold_thickness	two_hr_serum_insulin	bmi	diabetes_pedi	ag
0	0	6	92	62	32	126	32.0	0.085	4
1	0	5	132	80	0	0	26.8	0.186	6
2	0	3	106	72	0	0	25.8	0.207	2
3	0	4	99	68	38	0	32.8	0.145	3
4	0	4	96	56	17	49	20.8	0.340	2

In [8]:
```
print(df_train.head())
```

```
   diabetes_class  preg_count  glucose_concentration  diastolic_bp  \
0               0           6                     92            62   
1               0           5                    132            80   
2               0           3                    106            72   
3               0           4                     99            68   
4               0           4                     96            56   

   triceps_skin_fold_thickness  two_hr_serum_insulin   bmi  diabetes_pedi  age  
0                           32                   126  32.0          0.085   46  
1                            0                     0  26.8          0.186   69  
2                            0                     0  25.8          0.207   27  
3                           38                     0  32.8          0.145   33  
4                           17                    49  20.8          0.340   26  
```

In [9]:
```
df_validation.head()
```

Out[9]:

	diabetes_class	preg_count	glucose_concentration	diastolic_bp	triceps_skin_fold_thickness	two_hr_serum_insulin	bmi	diabetes_pedi	ag
0	0	1	130	70	13	105	25.9	0.472	2
1	1	8	133	72	0	0	32.9	0.270	3
2	0	0	137	68	14	148	24.8	0.143	2
3	0	2	88	74	19	53	29.0	0.229	2
4	1	9	130	70	0	0	34.2	0.652	4

In [10]:
```
print(df_validation.head())
```

```
   diabetes_class  preg_count  glucose_concentration  diastolic_bp  \
0               0           1                    130            70   
1               1           8                    133            72   
2               0           0                    137            68   
3               0           2                     88            74   
4               1           9                    130            70   

   triceps_skin_fold_thickness  two_hr_serum_insulin   bmi  diabetes_pedi  age  
0                           13                   105  25.9          0.472   22  
1                            0                     0  32.9          0.270   39  
2                           14                   148  24.8          0.143   21  
3                           19                    53  29.0          0.229   22  
4                            0                     0  34.2          0.652   45  
```

In [11]:
```
X_train = df_train.iloc[:,1:] # Features: 1st column onwards
y_train = df_train.iloc[:,0].ravel() # Target: 0th column

X_validation = df_validation.iloc[:,1:]
y_validation = df_validation.iloc[:,0].ravel()
```

In [12]:
```
# Launch a classifier
# XGBoost Training Parameter Reference:
#   https://xgboost.readthedocs.io/en/latest/parameter.html
classifier = xgb.XGBClassifier (objective="binary:logistic")
```

In [13]:
```
classifier
```

Out[13]:

▾ XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [41]:
```
print(classifier)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [14]:
```
classifier.fit(X_train,
               y_train,
               eval_set = [(X_train, y_train), (X_validation, y_validation)],
               eval_metric=['logloss'],
               early_stopping_rounds=10)
```

```
[0]	validation_0-logloss:0.55122	validation_1-logloss:0.60755
[1]	validation_0-logloss:0.46796	validation_1-logloss:0.57246
[2]	validation_0-logloss:0.40847	validation_1-logloss:0.54856
[3]	validation_0-logloss:0.36336	validation_1-logloss:0.53960
[4]	validation_0-logloss:0.32749	validation_1-logloss:0.53086
[5]	validation_0-logloss:0.30366	validation_1-logloss:0.52163
[6]	validation_0-logloss:0.28429	validation_1-logloss:0.52666
[7]	validation_0-logloss:0.26441	validation_1-logloss:0.52752
[8]	validation_0-logloss:0.24931	validation_1-logloss:0.52957
[9]	validation_0-logloss:0.23460	validation_1-logloss:0.52743
[10]	validation_0-logloss:0.22422	validation_1-logloss:0.53384
[11]	validation_0-logloss:0.21326	validation_1-logloss:0.53555
[12]	validation_0-logloss:0.20211	validation_1-logloss:0.54085
[13]	validation_0-logloss:0.19681	validation_1-logloss:0.54294
[14]	validation_0-logloss:0.19060	validation_1-logloss:0.54895
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/xgboost/sklearn.py:835: UserWarning: `eval_metric
` in `fit` method is deprecated for better compatibility with scikit-learn, use `eval_metric` in constructor or`set_
params` instead.
  warnings.warn(
/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/xgboost/sklearn.py:835: UserWarning: `early_stopp
ing_rounds` in `fit` method is deprecated for better compatibility with scikit-learn, use `early_stopping_rounds` in
constructor or`set_params` instead.
  warnings.warn(
```

Out[14]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
```

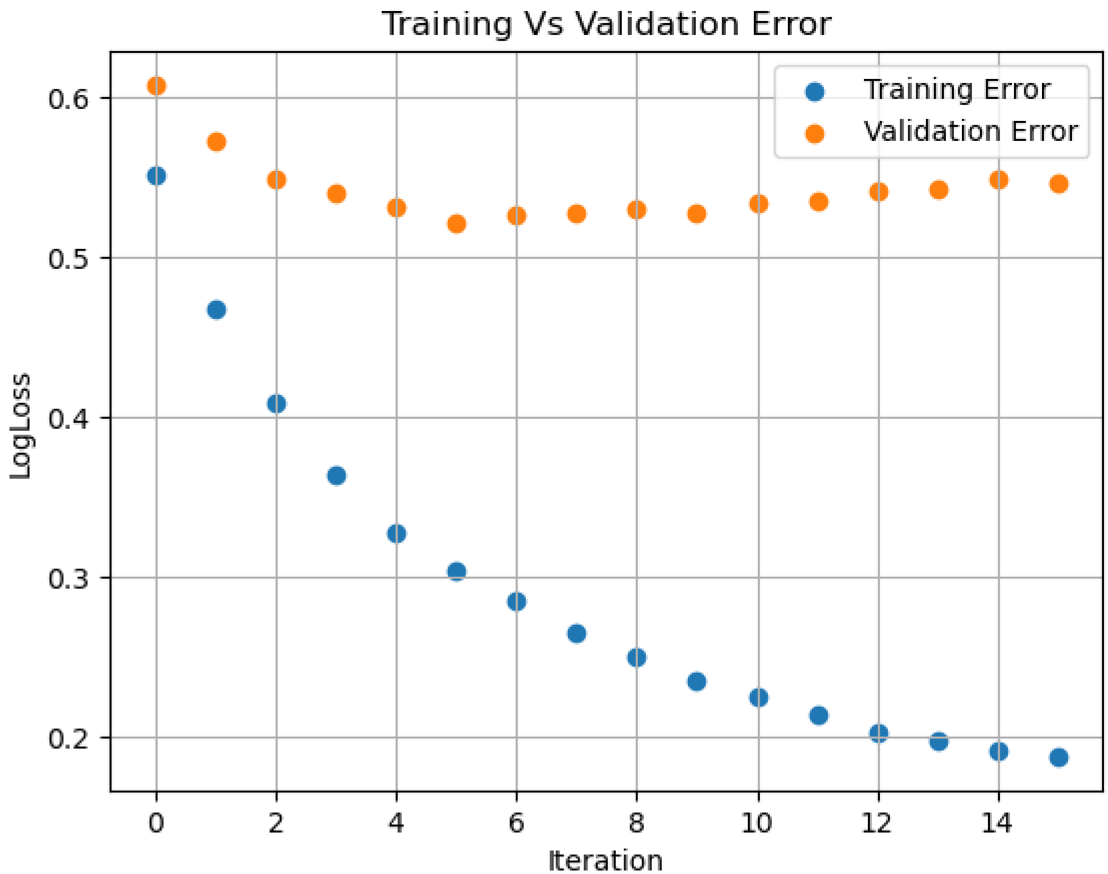
In [15]:
```
eval_result = classifier.evals_result()
```

In [16]:
```
training_rounds = range(len(eval_result['validation_0']['logloss']))
```

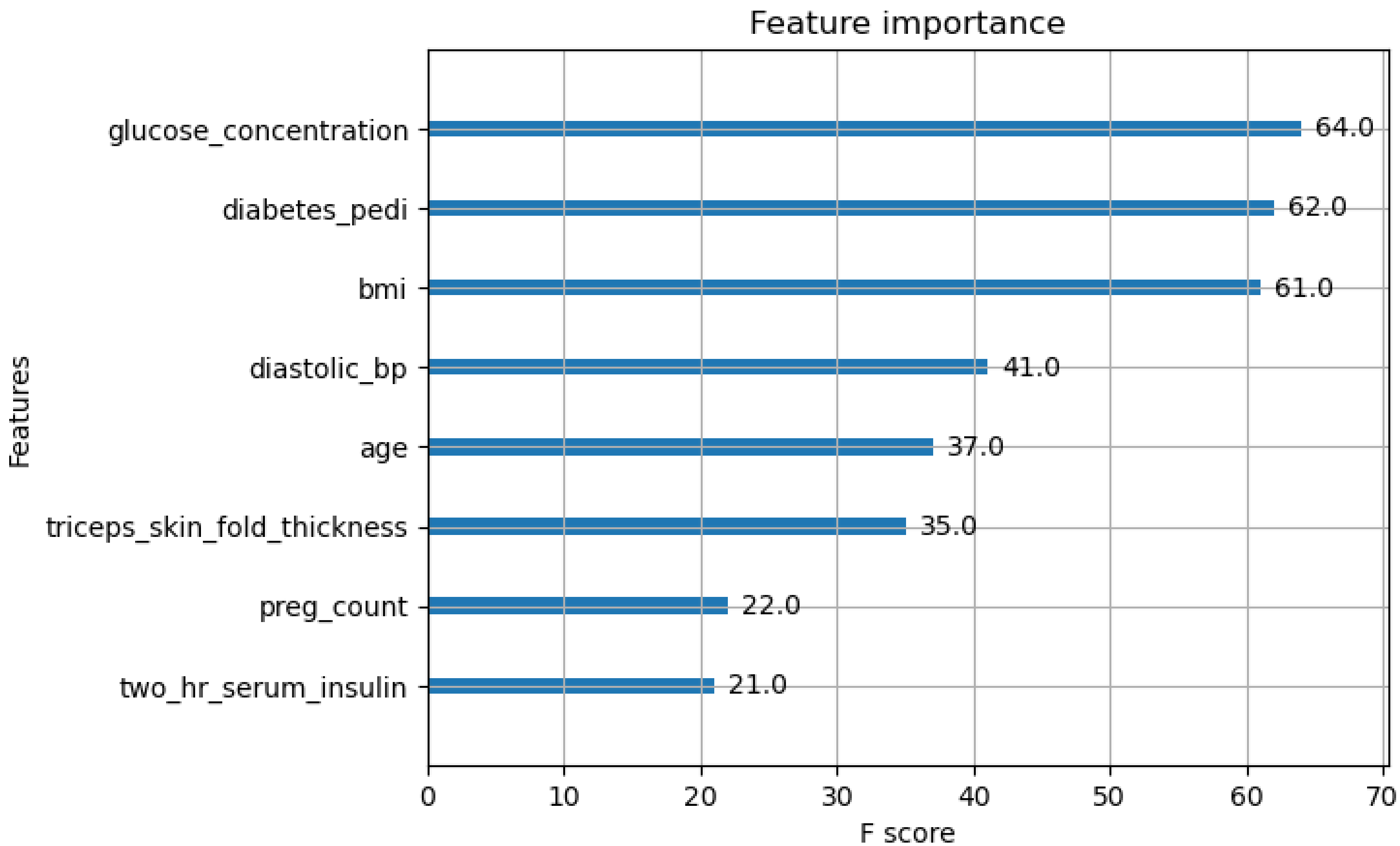
In [17]:
```
print(training_rounds)
```

```
range(0, 16)
```

In [18]:
```
plt.scatter(x=training_rounds,y=eval_result['validation_0']['logloss'],label='Training Error')
plt.scatter(x=training_rounds,y=eval_result['validation_1']['logloss'],label='Validation Error')
plt.grid(True)
plt.xlabel('Iteration')
plt.ylabel('LogLoss')
plt.title('Training Vs Validation Error')
plt.legend()
plt.show()
```



In [19]:
```
xgb.plot_importance(classifier)
plt.show()
```



In [20]:
```
df = pd.read_csv(validation_file,names=columns)
```

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

Out[21]:

	diabetes_class	preg_count	glucose_concentration	diastolic_bp	triceps_skin_fold_thickness	two_hr_serum_insulin	bmi	diabetes_pedi	ag
0	0	1	130	70	13	105	25.9	0.472	2
1	1	8	133	72	0	0	32.9	0.270	3
2	0	0	137	68	14	148	24.8	0.143	2
3	0	2	88	74	19	53	29.0	0.229	2
4	1	9	130	70	0	0	34.2	0.652	4

In [22]:
```
print(df.head())
```

```
   diabetes_class  preg_count  glucose_concentration  diastolic_bp  \
0               0           1                    130            70   
1               1           8                    133            72   
2               0           0                    137            68   
3               0           2                     88            74   
4               1           9                    130            70   

   triceps_skin_fold_thickness  two_hr_serum_insulin   bmi  diabetes_pedi  age  
0                           13                   105  25.9          0.472   22  
1                            0                     0  32.9          0.270   39  
2                           14                   148  24.8          0.143   21  
3                           19                    53  29.0          0.229   22  
4                            0                     0  34.2          0.652   45  
```

In [23]:
```
X_test = df.iloc[:,1:]
```

In [24]:
```
result = classifier.predict(X_test)
```

In [25]:
```
result[:5]
```

Out[25]:
```
array([0, 0, 0, 0, 1])
```

In [26]:
```
df['predicted_class'] = result
```

In [27]:
```
df.head()
```

Out[27]:

	diabetes_class	preg_count	glucose_concentration	diastolic_bp	triceps_skin_fold_thickness	two_hr_serum_insulin	bmi	diabetes_pedi	ag
0	0	1	130	70	13	105	25.9	0.472	2
1	1	8	133	72	0	0	32.9	0.270	3
2	0	0	137	68	14	148	24.8	0.143	2
3	0	2	88	74	19	53	29.0	0.229	2
4	1	9	130	70	0	0	34.2	0.652	4

In [28]:

```
print(df.head())
```

```
   diabetes_class  preg_count  glucose_concentration  diastolic_bp  \
0               0           1                    130            70   
1               1           8                    133            72   
2               0           0                    137            68   
3               0           2                     88            74   
4               1           9                    130            70   

   triceps_skin_fold_thickness  two_hr_serum_insulin   bmi  diabetes_pedi  \
0                           13                   105  25.9          0.472   
1                            0                     0  32.9          0.270   
2                           14                   148  24.8          0.143   
3                           19                    53  29.0          0.229   
4                            0                     0  34.2          0.652   

   age  predicted_class  
0   22                0  
1   39                0  
2   21                0  
3   22                0  
4   45                1  
```

Binary Classifier Metrics

In [29]:

```
# Reference: https://scikit-learn.org/stable/modules/model_evaluation.html
# Explicitly stating labels. Pass=1, Fail=0
def true_positive(y_true, y_pred):
```

```
    return confusion_matrix(y_true, y_pred,labels=[1,0])[0, 0]

def true_negative(y_true, y_pred):
    return confusion_matrix(y_true,y_pred,labels=[1,0])[1, 1]

def false_positive(y_true, y_pred):
    return confusion_matrix(y_true, y_pred,labels=[1,0])[1, 0]

def false_negative(y_true, y_pred):
    return confusion_matrix(y_true, y_pred,labels=[1,0])[0, 1]
```

In [30]:

```
# Compute Binary Classifier Metrics
# Returns a dictionary {"MetricName":Value,...}

def binary_classifier_metrics(y_true, y_pred):
    metrics = {}

    # References:
    #  https://docs.aws.amazon.com/machine-learning/latest/dg/binary-classification.html
    #  https://en.wikipedia.org/wiki/Confusion_matrix

    # Definition:
    # true positive = tp = how many samples were correctly classified as positive (count)
    # true negative = tn = how many samples were correctly classified as negative (count)
    # false positive = fp = how many negative samples were mis-classified as positive (count)
    # false_negative = fn = how many positive samples were mis-classified as negative (count)

    # positive = number of positive samples (count)
    #          = true positive + false negative
    # negative = number of negative samples (count)
    #          = true negative + false positive

    tp = true_positive(y_true, y_pred)
    tn = true_negative(y_true, y_pred)
    fp = false_positive(y_true, y_pred)
    fn = false_negative(y_true, y_pred)

    positive = tp + fn
    negative = tn + fp

    metrics['TruePositive'] = tp
    metrics['TrueNegative'] = tn
```

```
    metrics['FalsePositive'] = fp
    metrics['FalseNegative'] = fn

    metrics['Positive'] = positive
    metrics['Negative'] = negative

    # True Positive Rate (TPR, Recall) = true positive/positive
    # How many positives were correctly classified? (fraction)
    # Recall value closer to 1 is better. closer to 0 is worse
    if tp == 0:
        recall = 0
    else:
        recall = tp/positive

    metrics['Recall'] = recall

    # True Negative Rate = True Negative/negative
    # How many negatives were correctly classified? (fraction)
    # True Negative Rate value closer to 1 is better. closer to 0 is worse
    if tn == 0:
        tnr = 0
    else:
        tnr = tn/(negative)
    metrics['TrueNegativeRate'] = tnr

    # Precision = True Positive/(True Positive + False Positive)
    # How many positives classified by the algorithm are really positives? (fraction)
    # Precision value closer to 1 is better. closer to 0 is worse
    if tp == 0:
        precision = 0
    else:
        precision = tp/(tp + fp)
    metrics['Precision'] = precision

    # Accuracy = (True Positive + True Negative)/(total positive + total negative)
    # How many positives and negatives were correctly classified? (fraction)
    # Accuracy value closer to 1 is better. closer to 0 is worse
    accuracy = (tp + tn)/(positive + negative)
    metrics['Accuracy'] = accuracy

    # False Positive Rate (FPR, False Alarm) = False Positive/(total negative)
    # How many negatives were mis-classified as positives (fraction)
```

```
    # False Positive Rate value closer to 0 is better. closer to 1 is worse
    if fp == 0:
        fpr = 0
    else:
        fpr = fp/(negative)
    metrics['FalsePositiveRate'] = fpr

    # False Negative Rate (FNR, Misses) = False Negative/(total Positive)
    # How many positives were mis-classified as negative (fraction)
    # False Negative Rate value closer to 0 is better. closer to 1 is worse
    fnr = fn/(positive)
    metrics['FalseNegativeRate'] = fnr

    # F1 Score = harmonic mean of Precision and Recall
    # F1 Score closer to 1 is better. Closer to 0 is worse.
    if precision == 0 or recall == 0:
        f1 = 0
    else:
        f1 = 2*precision*recall/(precision+recall)

    metrics['F1'] = f1

    return metrics
```

In [31]:

```
# Reference:
# https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        #print("Normalized confusion matrix")
    #else:
    #    print('Confusion matrix, without normalization')

    #print(cm)
```

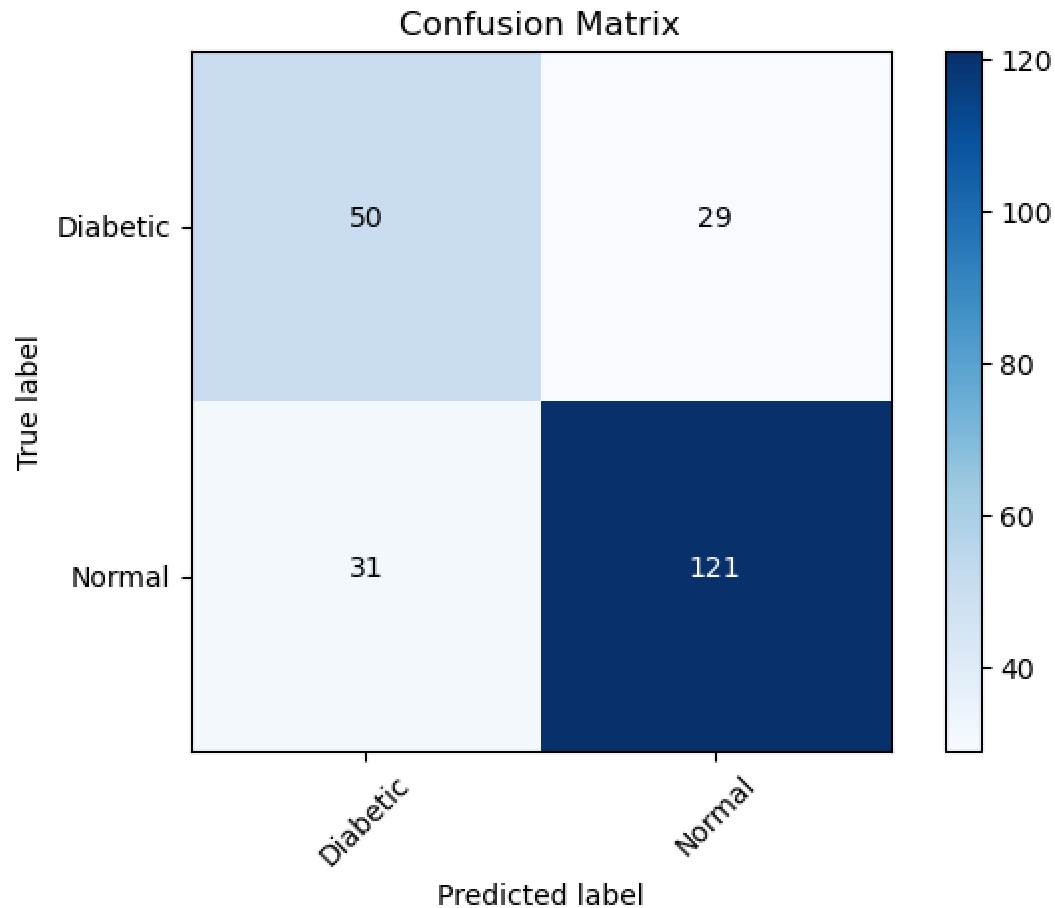
```
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

In [32]:

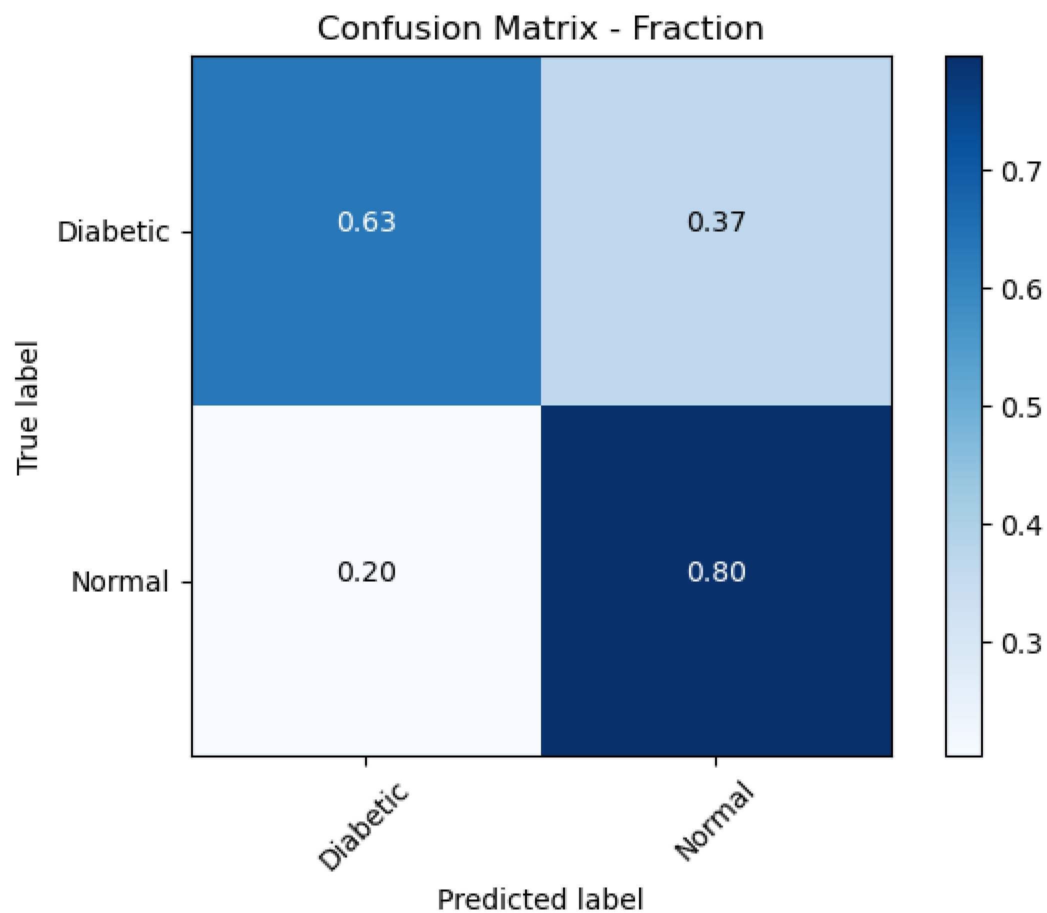
```
# Compute confusion matrix
cnf_matrix = confusion_matrix(df['diabetes_class'], df['predicted_class'],labels=[1,0])
```

In [33]:

```
# Plot confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['Diabetic','Normal'],
                      title='Confusion Matrix')
```



In [34]:
```
# Plot confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['Diabetic','Normal'],
                      title='Confusion Matrix - Fraction', normalize=True)
```



In [35]:
```
metrics = [binary_classifier_metrics(df['diabetes_class'], df['predicted_class'])]
df_metrics=pd.DataFrame.from_dict(metrics)
df_metrics.index = ['Model']
```

In [36]:
```
df_metrics
```

Out[36]:

	TruePositive	TrueNegative	FalsePositive	FalseNegative	Positive	Negative	Recall	TrueNegativeRate	Precision	Accuracy	False
Model	50	121	31	29	79	152	0.632911	0.796053	0.617284	0.74026	

In [37]:
```
print(df_metrics)
```

```
       TruePositive  TrueNegative  FalsePositive  FalseNegative  Positive  
Model            50           121             31             29        79   \

       Negative    Recall  TrueNegativeRate  Precision  Accuracy  
Model       152  0.632911          0.796053   0.617284   0.74026   \

       FalsePositiveRate  FalseNegativeRate     F1  
Model           0.203947           0.367089  0.625  
```

In [38]:

```
print('Counts')
print(df_metrics[['TruePositive',
                  'FalseNegative',
                  'FalsePositive',
                  'TrueNegative',]].round(2))
print()
print('Fractions')
print(df_metrics[['Recall',
                  'FalseNegativeRate',
                  'FalsePositiveRate',
                  'TrueNegativeRate',]].round(2))
print()

print(df_metrics[['Precision',
                  'Accuracy',
                  'F1']].round(2))
```

```
Counts
       TruePositive  FalseNegative  FalsePositive  TrueNegative
Model            50             29             31           121

Fractions
       Recall  FalseNegativeRate  FalsePositiveRate  TrueNegativeRate
Model    0.63               0.37                0.2               0.8

       Precision  Accuracy    F1
Model       0.62      0.74  0.63
```

In [39]:

```
print(classification_report(
    df['diabetes_class'],
    df['predicted_class'],
    labels=[1,0],
    target_names=['Diabetic','Normal']))
```

```
              precision    recall  f1-score   support

    Diabetic       0.62      0.63      0.63        79
      Normal       0.81      0.80      0.80       152

    accuracy                           0.74       231
   macro avg       0.71      0.71      0.71       231
weighted avg       0.74      0.74      0.74       231
```

In [40]:
```
# Yeah, not so good. Those dang zeros. We're going to fix it.
```

In [ ]: