Diabetes Prediction

December 22, 2022

1 Diabetes Prediction using ML

1.1 Importing important libraries

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC                                    #importing support␣
 ↪vector regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix , classification_report
from sklearn.pipeline import make_pipeline
import pickle
%matplotlib inline
```

2 Data collection and analysis

2.1 Pima dataset

[2]:
```
db_df = pd.read_csv('diabetes.csv')
```

[3]:
```
#Finding the number of Number of rows and colmns of dataset
db_df.shape
```

[3]:
```
(768, 9)
```

[4]:
```
#Looking the top 5 data of dataset
db_df.head()
```

[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	

```
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[5]:
```
#Getting the statistical measure of data
db_df.describe()
```

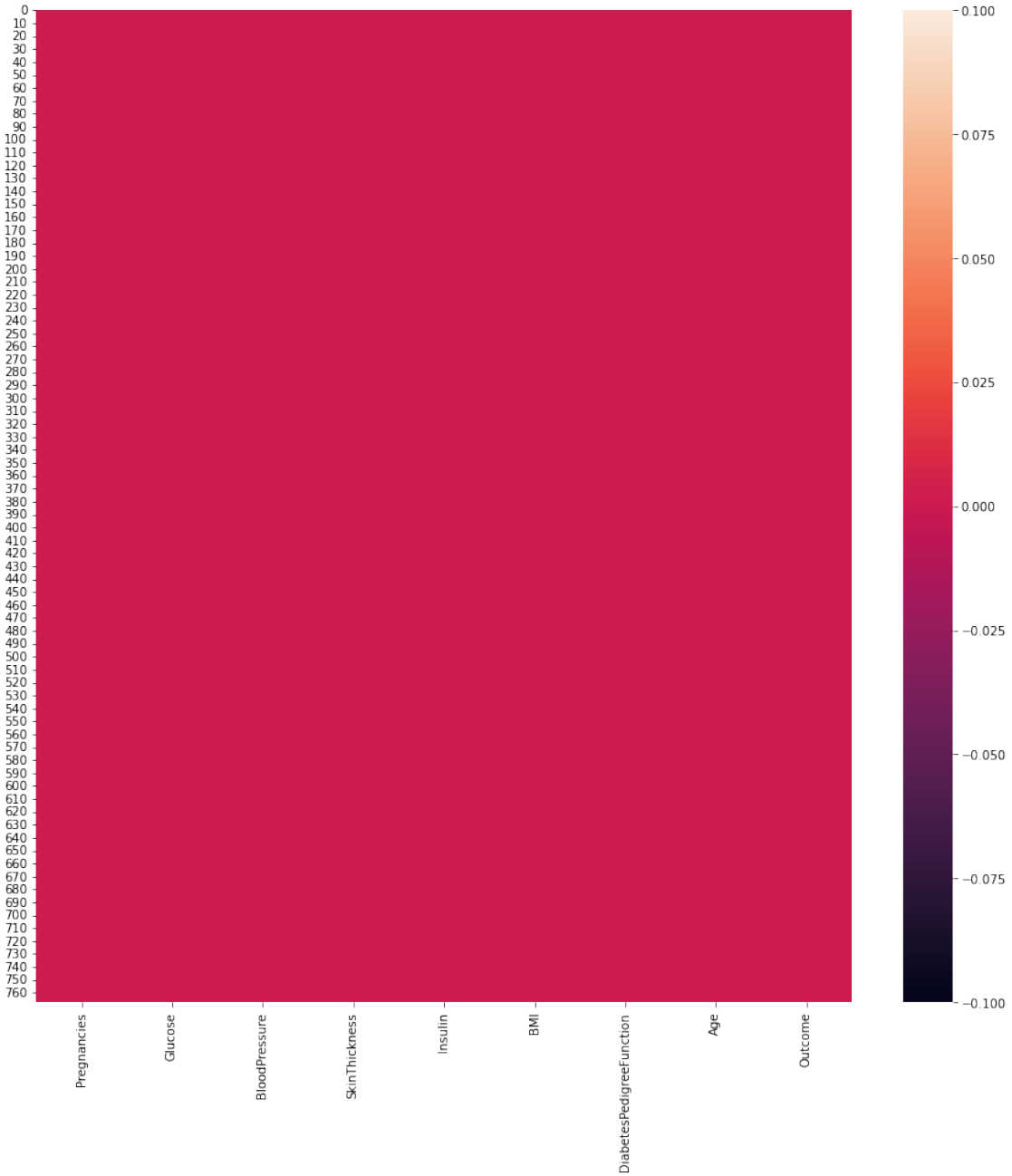
[5]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

[6]:
```
#cheching for empty value in dataset
db_df.isnull().sum()
```

[6]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[7]:
```
#plotting a heatmap to check for null value
plt.figure(figsize=(15,15))
sn.heatmap(db_df.isnull()) #since we get heatmap of uniform colour so no data␣
 ↪is missing
```

[7]: <AxesSubplot:>



[8]:
```
#checking data in dataset for true or false data
db_df_shpe = db_df['Outcome'].value_counts()
print('The total data not having diabetes:-{}\nThe total data having diabetes:
 ↪-{}'.format(db_df_shpe[0],db_df_shpe[1]))
```

```
The total data not having diabetes:-500
The total data having diabetes:-268
```

[9]:
```
#Getting more analysis of our data
db_df.groupby('Outcome').mean()
```

[9]:

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.190000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.067164

[10]:
```
#seprating the data and labels
X = db_df.drop(['Outcome'],axis=1)
y = db_df['Outcome']
```

[11]:
```
X
```

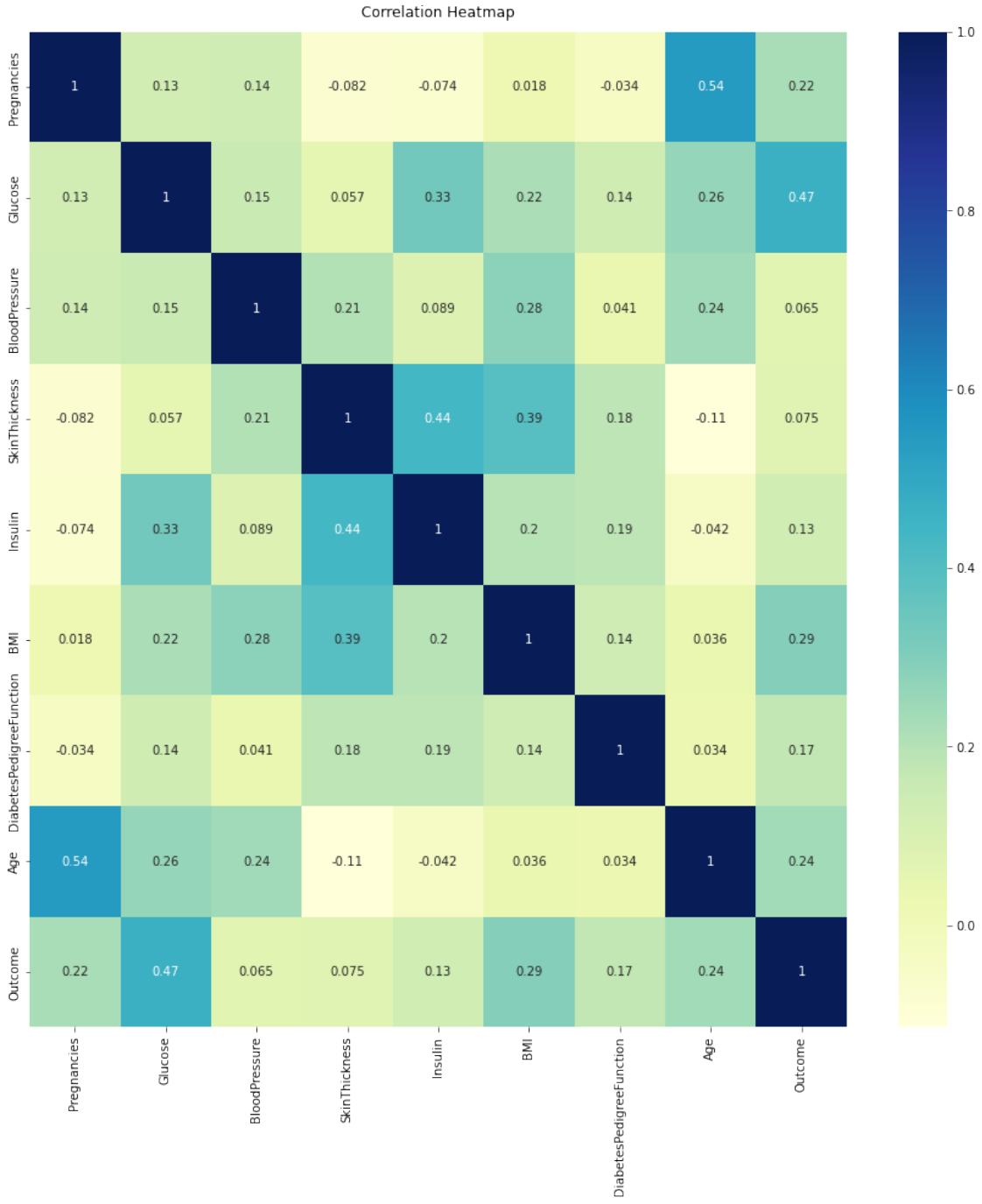
[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
..	…	…	…	…	…	…	…	…
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8		
765	5	121	72	23	112	26.2		
766	1	126	60	0	0	30.1		
767	1	93	70	31	0	30.4		

```
764                    0.340   27
765                    0.245   30
766                    0.349   47
767                    0.315   23

[768 rows x 8 columns]
```

3 Creating a correlational matrix

[12]:
```
plt.figure(figsize=(15,15))
heatmap = sn.heatmap(db_df.corr(),annot=True,cmap="YlGnBu");
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':12}, pad=12);
```



3.1 Data Standardization

[13]:
```
scaler =StandardScaler()
X_standardised = scaler.fit_transform(X)
```

[14]:
```
X_standardised
```

[14]:
```
array([[ 0.63994726,  0.84832379,  0.14964075, …,  0.20401277,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575, …, -0.68442195,
        -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, …, -1.10325546,
         0.60439732, -0.10558415],
       …,
       [ 0.3429808 ,  0.00330087,  0.14964075, …, -0.73518964,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.1597866 , -0.47073225, …, -0.24020459,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.8730192 ,  0.04624525, …, -0.20212881,
        -0.47378505, -0.87137393]])
```

3.2 Train test split

[15]:
```
X_train ,X_test,y_train,y_test = train_test_split(X_standardised,y,test_size=0.
↪2,random_state=12,stratify=y)
```

[16]:
```
print('The original data shape is {}. Test data shape {} and train data shape␣
↪is {}'.format(X.shape,X_train.shape,X_test.shape))
```

```
The original data shape is (768, 8). Test data shape (614, 8) and train data
shape is (154, 8)
```

3.3 Selecting the best possible model with hyperperameters between logestic_regression and support vector classifier.

[17]:
```
model_params = {
    'logestic_regression' :{
        'model' : LogisticRegression(),
        'params' :{
            'penalty':['l1', 'l2', 'elasticnet', None],
            'C':[-7,1e-2,0,1,2,3,4,5,6,7,8,9,10,20,30,40,50],
            'max_iter':[10,50,100,200,300,500],
            'tol':[1e-5,1e-4,1e-6,1e-8]
     }
  },
    'SVC':{
     'model' : SVC(),
     'params':{
         'gamma':['auto','scale'],
         'C':[-7,1e-2,0,1,2,3,4,5,6,7,8,9,10,20,30,40,50],
         'kernel':['linear', 'poly', 'rbf', 'sigmoid'],
         'coef0':[0.0,0.5,0.7,0.9,1.0,2.0]
     }
   }
```

```
}
```

[18]:
```
scores = []
for model_name,mp in model_params.items():
    clf = GridSearchCV(mp['model'],mp['params'],cv =5,return_train_score=False)
    clf.fit(X_train,y_train)
    scores.append({
        'model':model_name,
        'best_score':clf.best_score_,
        'best_params':clf.best_params_,
    })
```

```
C:\Users\01abn\anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
C:\Users\01abn\anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
C:\Users\01abn\anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
C:\Users\01abn\anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
```

```
 0.7100893  0.67093163 0.76699987 0.70194589 0.70846328 0.64647474
 0.76699987 0.70194589 0.7100893  0.6497401  0.76699987 0.69706784
 0.70846328 0.65621751 0.76699987 0.69706784 0.7100893  0.63833133
 0.76699987 0.70031987 0.70846328 0.66613355 0.76699987 0.69869386
 0.7100893  0.66775956 0.76699987 0.69544182 0.70846328 0.66775956
 0.76699987 0.69706784 0.7100893  0.66450753 0.76699987 0.69706784
 0.70846328 0.65631081 0.76699987 0.69869386 0.7100893  0.64167666
 0.76699987 0.69540184 0.71985872 0.67255764 0.76699987 0.6937625
 0.71660669 0.6709183  0.76699987 0.69541517 0.71985872 0.64484873
 0.76699987 0.69541517 0.71660669 0.64484873 0.76699987 0.69540184
 0.71985872 0.65133946 0.76699987 0.69540184 0.71660669 0.67098494
 0.76699987 0.69702786 0.71985872 0.66775956 0.76699987 0.69702786
 0.71660669 0.66448087 0.76699987 0.70027989 0.71985872 0.65312542
 0.76699987 0.69702786 0.71660669 0.65475143 0.76699987 0.69868053
 0.71985872 0.63681194 0.76699987 0.6986672  0.71660669 0.63843796]
  warnings.warn(
```

[19]:
```
t_df = pd.DataFrame(scores,columns=['model','best_score','best_params'])
```

[20]:
```
print(t_df)
```

```
                 model  best_score  \
0  logestic_regression    0.767013
1                  SVC    0.768626

                                          best_params
0  {'C': 1, 'max_iter': 10, 'penalty': 'l2', 'tol…
1  {'C': 1, 'coef0': 0.0, 'gamma': 'auto', 'kerne…
```

[31]:
```
clf.best_score_
```

[31]:
```
0.7686258829801413
```

4 Therefore on comparing the perfomance of both the model we conlude that svc work better

4.0.1 Model evalution

Accuracy score

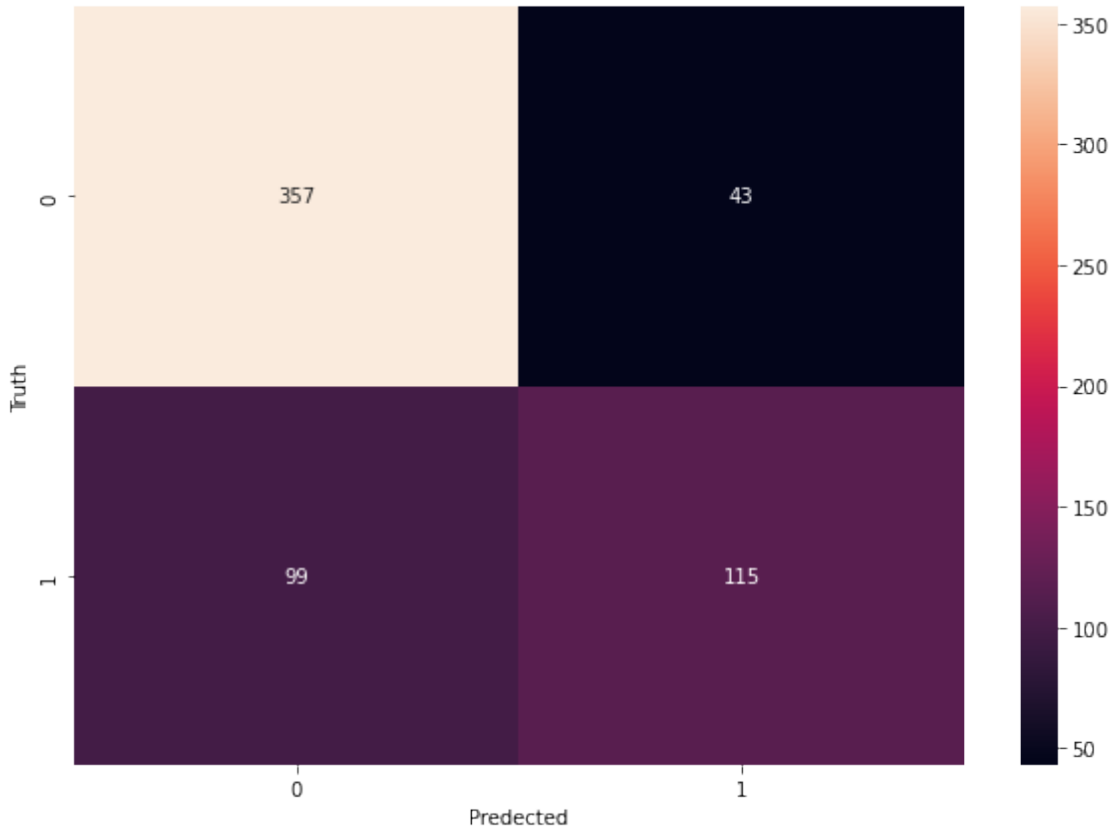
[21]:
```
y_test_predict = clf.predict(X_test)
y_train_predict = clf.predict(X_train)
res1= accuracy_score(y_test_predict,y_test)
res2 = accuracy_score(y_train_predict,y_train)
```

[22]:
```
print('Accuracy score on test is {} and train data for our model is {}'.
  ↪format(res1,res2))
```

```
Accuracy score on test is 0.7857142857142857 and train data for our model is
0.7687296416938111
```

[23]:
```
#creating the confusion matrix for train data checking the accuracy of our model
cm= confusion_matrix(y_true = y_train,y_pred=y_train_predict)

plt.figure(figsize=(10,7))
sn.heatmap(cm,annot=True,fmt='d')
plt.xlabel('Predected')
plt.ylabel('Truth')
```

[23]:
```
Text(69.0, 0.5, 'Truth')
```



4.1 Making a predictive system

[29]:
```
input_data = (5,187,76,27,207,43.6,1.034,53)

# changing the input_data to numpy array
input_data = np.asarray(input_data)


# reshape the array as we are predicting for one instance
input_data = input_data.reshape(1,-1)
```

```
#standarised the data
input_data_std = scaler.transform(input_data)
print(input_data_std)

prediction = clf.predict(input_data_std)
if prediction[0] == 1:
    print('The patient has Diabetes')
else:
    print('The patient has not Diabatese')
```

```
[[0.3429808  2.06891246 0.35643175 0.40544544 1.10446888 1.47320522
  1.69768028 1.68125866]]
The patient has Diabetes
```

```
C:\Users\01abn\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X
does not have valid feature names, but StandardScaler was fitted with feature
names
  warnings.warn(
```

4.2 Saving the model

[25]:
```
#Saving the scaler and model
filename = 'diabetes_model.sav'
pickle.dump(clf, open(filename, 'wb'))
scalerfile = 'scaler.sav'
pickle.dump(scaler, open(scalerfile, 'wb'))
```

[26]:
```
# loading the saved model
load_model = pickle.load(open('diabetes_model.sav', 'rb'))
load_scaler = pickle.load(open('scaler.sav','rb'))
```

[40]:
```
input_data = (5,104,74,0,0,28.8,0.153,48)

# changing the input_data to numpy array
input_data = np.asarray(input_data)


# reshape the array as we are predicting for one instance
input_data = input_data.reshape(1,-1)

#standarised the data
print('before input data',input_data)
input_data_std = load_scaler.transform(input_data)
print('after std input data',input_data_std)
print()
prediction = clf.predict(input_data_std)
if prediction[0] == 1:
```

```
    print('The patient has Diabetes')
else:
    print('The patient has not Diabatese')
```

```
before input data [[  5.    104.     74.      0.      0.     28.8     0.153  48.
]]
after std input data [[ 0.3429808  -0.5287506   0.25303625 -1.28821221
-0.69289057 -0.40519961
  -0.96304428  1.2558199 ]]

The patient has not Diabatese

C:\Users\01abn\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X
does not have valid feature names, but StandardScaler was fitted with feature
names
  warnings.warn(
```

[38]:
```
for col in db_df:
    print(col)
```

```
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
```

[ ]:

[ ]: