Ensemble_Learning

June 9, 2023

1 Problem Statements: Diabetes Prediction

[ ]:
```
#Let's start with importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
#read the data file
#data = pd.read_csv("/config/workspace/Dataset/diabetes.csv")
data=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/DS_PROJECT/
 ↪Diabetes_Prediction/Dataset/diabetes.csv")
data.head()
```

[ ]:
```
   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
```

[ ]:
```
data.describe()
```

[ ]:
```
       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
```

[ ]:
```
data.isnull().sum()
```

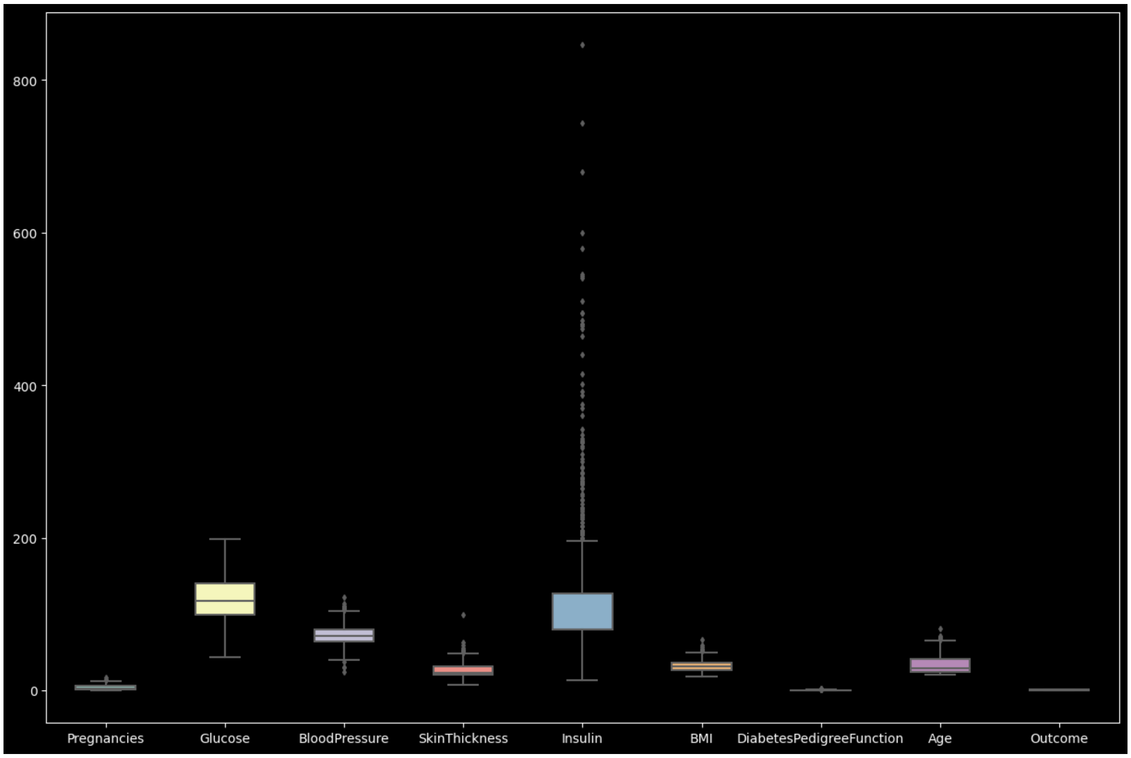
[ ]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

We can see there few data for columns Glucose , Insulin, skin thickenss, BMI and Blood Pressure which have value as 0. That's not possible,right? you can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace it with their respective mean values. Let's do the latter.

[ ]:
```
#here few misconception is there lke BMI can not be zero, BP can't be zero,␣
 ↪glucose, insuline can't be zero so lets try to fix it
# now replacing zero values with the mean of the column
data['BMI'] = data['BMI'].replace(0,data['BMI'].mean())
data['BloodPressure'] = data['BloodPressure'].replace(0,data['BloodPressure'].
 ↪mean())
data['Glucose'] = data['Glucose'].replace(0,data['Glucose'].mean())
data['Insulin'] = data['Insulin'].replace(0,data['Insulin'].mean())
data['SkinThickness'] = data['SkinThickness'].replace(0,data['SkinThickness'].
 ↪mean())
```

[ ]:
```
#now we have dealt with the 0 values and data looks better. But, there still are outliers present in some columns.lets visualize it
plt.style.use('dark_background')
fig, ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=data, width= 0.5,ax=ax,  fliersize=3)
```

[ ]:
```
<Axes: >
```



[ ]:
```
data.head()
```

[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148.0	72.0	35.000000	79.799479	33.6	
1	1	85.0	66.0	29.000000	79.799479	26.6	
2	8	183.0	64.0	20.536458	79.799479	23.3	
3	1	89.0	66.0	23.000000	94.000000	28.1	
4	0	137.0	40.0	35.000000	168.000000	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

```
[ ]: #segregate the dependent and independent variable
X = data.drop(columns = ['Outcome'])
y = data['Outcome']
```

```
[ ]: # separate dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪25,random_state=0)
X_train.shape, X_test.shape
```

```
[ ]: ((576, 8), (192, 8))
```

2 Ensemble Technique

- **Ensemble learning is a machine learning technique that combines multiple models to create a more accurate and robust model than any of the individual models could achieve on its own. Ensemble methods are often used when the data is noisy or when the underlying relationship between the features and the target variable is complex.**

3 Simple Ensemble Techniques

In this section, we will look at a few simple but powerful techniques, namely:

1. Max Voting
2. Averaging
3. Weighted Averaging

4 1. Max Voting

- **The max voting method is generally used for classification problems. In this technique, multiple models are used to make predictions for each data point. The predictions by each model are considered as a 'vote'. The predictions which we get from the majority of the models are used as the final prediction.**

let's see how well our model performs on the test data set.

```
[ ]: import warnings
warnings.filterwarnings("ignore")
```

```
[ ]: from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
model1 = LogisticRegression(random_state=1)
model2 = DecisionTreeClassifier(random_state=1)
model = VotingClassifier(estimators=[('lr', model1), ('dt', model2)],␣
 ↪voting='hard')
model.fit(X_train,y_train)
model.score(X_test,y_test)
```

[ ]: 0.7864583333333334

accuracy = accuracy_score(y_test,y_pred) accuracy

5 2. Averaging

- **Similar to the max voting technique, multiple predictions are made for each data point in averaging. In this method, we take an average of predictions from all the models and use it to make the final prediction. Averaging can be used for making predictions in regression problems or while calculating probabilities for classification problems.**

[ ]:
```
from sklearn.neighbors import KNeighborsClassifier
model1 = DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()

model1.fit(X_train,y_train)
model2.fit(X_train,y_train)
model3.fit(X_train,y_train)

pred1=model1.predict_proba(X_test)
pred2=model2.predict_proba(X_test)
pred3=model3.predict_proba(X_test)

finalpred=(pred1+pred2+pred3)/3
```

6 3. Weighted Average

- **This is an extension of the averaging method. All models are assigned different weights defining the importance of each model for prediction. For instance, if two of your colleagues are critics, while others have no prior experience in this field, then the answers by these two friends are given more importance as compared to the other people.**

[ ]:
```
model1 = DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()

model1.fit(X_train,y_train)
model2.fit(X_train,y_train)
model3.fit(X_train,y_train)

pred1=model1.predict_proba(X_test)
pred2=model2.predict_proba(X_test)
pred3=model3.predict_proba(X_test)
```

```
finalpred=(pred1*0.3+pred2*0.3+pred3*0.4)
```