W.3: Classification

1. First, if your data does not contain a categorical output variable, turn your continuous output variable into a binary variable. For instance, if your output variable was house price, $Y=0$ if $Y\lt 500,000$ and $Y=1$ otherwise. If your dataset has another output variable that is categorical, feel free to use this variable

- **Dummy variables were created for multinomial predictor variables and the variable 'NObeyesdad' is used as a multinomial categorical variable to predict. The unique values are shown below.**

```
['Insufficient_Weight' 'Normal_Weight' 'Obesity_Type_I' 'Obesity_Type_II'
 'Obesity_Type_III' 'Overweight_Level_I' 'Overweight_Level_II']
```

2. Randomly choose 80% of your data as the training set. The rest 20% of the data will be your test set.

- **The classes shown above are distributed unevenly. This being considered, a stratified random sample was taken and split accordingly.**

3. Apply the classification methods we learned in class on your training dataset and compare their misclassification rates on the test dataset.

- **The results shown below are taken from predictions on the test set of the various models.**

```
Logistic regression:
Confusion Matrix:
[[51  3  0  0  0  0  0]
 [ 9 38  0  0  0 10  1]
 [ 0  0 54  5  2  1  8]
 [ 0  0  0 59  1  0  0]
 [ 0  0  0  1 64  0  0]
 [ 0  7  4  0  0 38  9]
 [ 0  0 12  0  1 12 33]]
Error rate: 0.20330969267139476
```

```
Naive Bayes:
Confusion Matrix:
[[51  2  1  0  0  0  0]
 [34 17  3  0  0  2  2]
 [ 0  4 33 29  0  0  4]
 [ 0  0  3 55  0  0  2]
 [ 0  0  0  0 64  0  1]
 [15  5 20  8  0  1  9]
 [ 6  2 26 11  0  1 12]]
Error rate: 0.4491725768321513

LDA:
Confusion Matrix:
[[52  2  0  0  0  0  0]
 [11 39  0  0  0  7  1]
 [ 0  0 68  1  0  0  1]
 [ 0  0  0 60  0  0  0]
 [ 0  0  0  0 65  0  0]
 [ 0  7  0  0  0 43  8]
 [ 0  0  3  0  0  4 51]]
Error rate: 0.1063829787234043
```

```
QDA:
Confusion Matrix:
[[50  4  0  0  0  0  0]
 [11 40  0  0  0  6  1]
 [ 0  6 53  8  0  0  3]
 [ 0  0  2 58  0  0  0]
 [ 0  0  1  0 64  0  0]
 [ 0  7  2  0  0 38 11]
 [ 0  5  3  0  0  1 49]]
Error rate: 0.1678486997635934

SVM:
Confusion Matrix:
[[53  1  0  0  0  0  0]
 [ 9 42  0  0  0  6  1]
 [ 0  0 60  5  0  0  5]
 [ 0  0  1 59  0  0  0]
 [ 0  0  0  1 64  0  0]
 [ 0  7  1  0  0 46  4]
 [ 0  0 10  0  0  7 41]]
Error rate: 0.1371158392434988
```

4. Provide interpretation. Which method gave you the best result? Based on what we have learned in class, why do you think this method worked the best for your data?

- **LDA performed the best as it had the lowest error rate amongst any of the models and the best looking confusion matrix. This makes sense as LDA is often very practical for multinomial prediction. SVM with a linear kernel provided a close second. The high performance of both of these models indicate that the data reacts well to models that utilize linear separation approaches.**

Python Code

```
#importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn import svm
from sklearn.metrics import accuracy_score

#setting seed
np.random.seed(100)

#importing credit data as a pandas dataframe
obesity = pd.read_csv('ObesityDataSet_raw_and_data_sinthetic.csv')

#setting up dummy variables for categorical variables
Gender = pd.get_dummies(obesity['Gender'])
History = pd.get_dummies(obesity['family_history_with_overweight'])
FAVC = pd.get_dummies(obesity['FAVC'])
CAEC = pd.get_dummies(obesity['CAEC'])
SMOKE = pd.get_dummies(obesity['SMOKE'])
SCC = pd.get_dummies(obesity['SCC'])
CALC = pd.get_dummies(obesity['CALC'])
MTRANS = pd.get_dummies(obesity['MTRANS'])
obesity = pd.concat([obesity, Gender, History, FAVC, CAEC, SMOKE, SCC, CALC, MTRANS],
axis = 1)
obesity.drop(['Gender', 'Male', 'family_history_with_overweight', 'no', 'FAVC', 'CAEC', 'SMOKE',
'SCC', 'CALC', 'MTRANS', 'Walking'], inplace=True, axis=1)

#extracting a stratified random sample for train and test data
X_train, X_test, y_train, y_test = train_test_split(obesity.loc[:, obesity.columns !='NObeyesdad'],
obesity['NObeyesdad'], test_size=0.2, stratify=obesity['NObeyesdad'])

#setting up logistic regression model
model = LogisticRegression(solver='lbfgs', multi_class = 'multinomial', max_iter=1000)

#fitting model with data
model.fit(X_train, y_train)
```

```
#printing confusion matrix of model
print('Confusion Matrix:')
print(confusion_matrix(y_test, model.predict(X_test)))

#printing accuracy rate of predictions
print(f'Accuracy rate: {accuracy_score(y_test, model.predict(X_test))}')

#setting up naive bayes model
model = GaussianNB()

#training model
model.fit(X_train, y_train)

#printing confusion matrix of model
print('Confusion Matrix:')
print(confusion_matrix(y_test, model.predict(X_test)))

#printing accuracy rate of predictions
print(f'Accuracy rate: {accuracy_score(y_test, model.predict(X_test))}')

#setting up naive bayes model
model = LinearDiscriminantAnalysis()

#training model
model.fit(X_train, y_train)

#printing confusion matrix of model
print('Confusion Matrix:')
print(confusion_matrix(y_test, model.predict(X_test)))

#printing accuracy rate of predictions
print(f'Accuracy rate: {accuracy_score(y_test, model.predict(X_test))}')

#setting up naive bayes model
model = QuadraticDiscriminantAnalysis(reg_param=0.001)

#training model
model.fit(X_train, y_train)

#printing confusion matrix of model
print('Confusion Matrix:')
print(confusion_matrix(y_test, model.predict(X_test)))

#printing accuracy rate of predictions
```

```
print(f'Accuracy rate: {accuracy_score(y_test, model.predict(X_test))}')

#Create a svm Classifier
model = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
model.fit(X_train, y_train)

#printing confusion matrix of model
print('Confusion Matrix:')
print(confusion_matrix(y_test, model.predict(X_test)))

#printing accuracy rate of predictions
print(f'Accuracy rate: {accuracy_score(y_test, model.predict(X_test))}')
```