ELG5255 Applied Machine Learning

Group Assignment 2

Group 4:

1. Yomna Mohamed Sayed Ahmed ID: 300327217
2. Khaled Mohamed Mohamed Mahmoud ID: 300327242
3. Marwa Hamdi Boraie Mahmoud ID: 300327267

Part 1: Calculations

- Using Bayesian Rule Based Classifier to make prediction when Color = G, Gender = F, Price=H . Please include the detailed calculation process.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Color(x1)	R	R	R	R	R	G	G	G	G	G	Y	Y	Y	Y	Y
Gender(x2)	M	M	F	F	M	M	M	M	F	F	F	F	M	M	F
Price(x3)	H	L	L	H	M	M	H	L	L	M	L	H	M	L	M
TARGET(y)	N	N	Y	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y	N

1. Calculate the probability of TARGET (y) for Yes and No labels:
   →P(TARGET= Y) = 9/15
   →P(TARGET= N) = 6/15

2. Calculate the probability of Color(x1) , Gender(x2) ,and Price(x3) given the value of the TARGET(y) which is Yes or No.

Color(x1)	Y	N
R	2/9	3/6
G	3/9	2/6
Y	4/9	1/6

Price(x3)	Y	N
H	2/9	2/6
M	2/9	3/6
L	5/9	1/6

Gender(x2)	Y	N
M	3/9	5/6
F	6/9	1/6

3. Using Bayesian Rule to make prediction, assume this condition (Color = G, Gender = F, Price=H) = New Instance

   →P(Yes|New Instance) = P(Yes)*P(Color=G|Yes)*P(Gender=F|Yes)
   *P(Price=H|Yes)
   =(9/15)*(3/9)*(6/9)*(2/9)= 4/135

   → P(No|New Instance) = P(No)*P(Color=G|No)*P(Gender=F|No)
   *P(Price=H|No)
   =(6/15)*(2/6)*(1/6)*(2/6)= 1/135

4. Normalize these values by using these equations:

- $P(Yes|New\ Instance) = \frac{P(Yes|New\ Instance)}{P(Yes|New\ Instance)+P(No|New\ Instance)}$ = 4/5 = 0.8
- $P(No|New\ Instance) = \frac{P(No|New\ Instance)}{P(Yes|New\ Instance)+P(No|New\ Instance)}$ = 1/5 = 0.2

5. ∵ P(Yes|New Instance) > P(No|New Instance)

   ∴ The prediction of the TARGET(y) will be Yes.

- Consider the following loss table, which contains three actions and two classes. Calculate the expected risk of three actions, and determine the rejection area of P(Class1| x).

Target	Class 1	Class 2
a1(choose class 1)	0	5
a2(choose class 2)	5	2
a3 (Rejection)	4	4

$\lambda_{11} = 0$, $\lambda_{12} = \lambda_{21}$ 5 , $\lambda_{22} = 2$ P(class1|x) + P(class2|x) = 1

P(class2|x)=1- P(class1|x)

1. Calculate the expected risk of three actions, and determine the rejection area of P(Class1| x):

   →R(α1|x) = λ_{11} * P(class1|x)+ λ_{12}* P(class2|x)

   =0 *P(class1|x)+*5 P(class2|x)

   =5*(1- P(class1|x))

   =5-5* P(class1|x)   ---------------------> (1)

→R(α2|x)= λ_{21}* P(class1|x)+ λ_{22}* P(class2|x)

=5* P(class1|x)+2* P(class2|x)

=5*P(class1|x)+2*(1- P(class1|x)

=3 *P(class1|x)+2 ------------------------ > (2)

→R(α3|x) = 4 ---------------------------- > (3)

- From 1 we choose α1 if :

5-5* P(class1|x) < 4

P(class1|x) > 1/5

- From 1 we choose α2 if :

3*P(class1|x)+2 < 4

3*P(class1|x) < 2

P(class1|x) < 2/3

→*We reject otherwise, that is, if 2/3 < P(*Class1|x*) < 1 /5*

∵ *There is no intersection between P(*Class1|x*) < 1 /5 and P(*Class1|x*) > 2/3*

∴ *The* Rejection Area of P(Class1| x) = None.

Part 2: Programming

(1) Naive Bayesian classifier (GaussianNB)

Loading Data:

Loading Data

```
[6] from sklearn.datasets import load_wine
    data = load_wine()
```

a) Using train test split function in scikitlearn to split the dataset into a training set, a testing set.

Spliting Data

```
# Import train_test_split function
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, test_size=0.3,random_state=1) # 70% training and 30% test
```

Using Naïve Bayesian classifier (GaussianNB)to Train the model

```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
model = GaussianNB()

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

#Predict the response for test dataset
y_pred = model.predict(X_test)
```

Getting_Acurrecy=98%

Getting Accuracy

```
[ ] #Import scikit-learn metrics module for accuracy calculation
    from sklearn import metrics

    # Model Accuracy, how often is the classifier correct?
    print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9814814814814815
```

b) Using classification report function for calculating precision, recall and f1.

Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred , target_names=data.target_names))
```

```
              precision    recall  f1-score   support

     class_0       0.96      1.00      0.98        23
     class_1       1.00      0.95      0.97        19
     class_2       1.00      1.00      1.00        12

    accuracy                           0.98        54
   macro avg       0.99      0.98      0.98        54
weighted avg       0.98      0.98      0.98        54
```

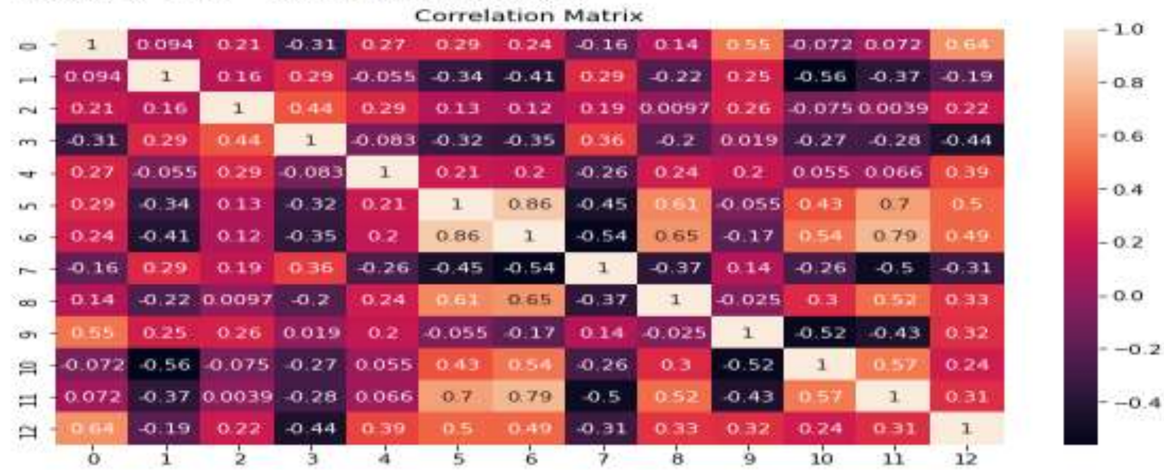
Feature Selection:

- **Corrleation Matrix**

```
cor = our_data.corr()

plt.figure(figsize= (10,6))
sns.heatmap(cor, annot = True)
plt.title("Correlation Matrix")
```

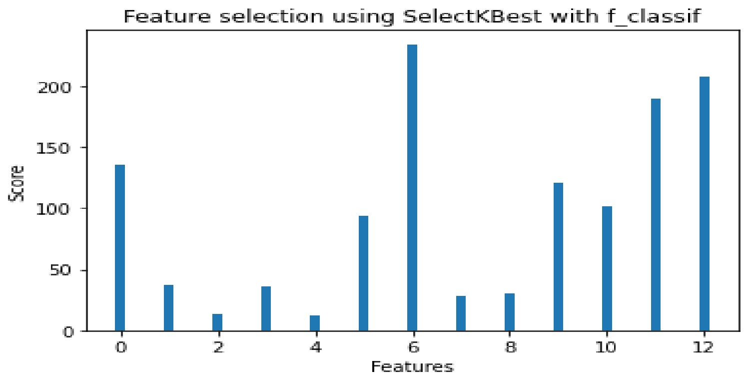
```
Text(0.5, 1.0, 'Correlation Matrix')
```



-By Using `SelectKBest,f_classif Fuctions in` scikitlearn to choose the two most influential in the data

```
from sklearn.feature_selection import SelectKBest,f_classif
featureSelection= SelectKBest(f_classif, k=2).fit(our_data, our_target)
X_train = featureSelection.transform(our_data)
print(featureSelection.get_feature_names_out())
featureSelection.scores_
col_names =our_data.columns
print(f"accuracies :{featureSelection.scores_}")
plt.bar(col_names , featureSelection.scores_, width=0.2)
plt.title("Feature selection using SelectKBest with f_classif")
plt.xlabel("Features")
plt.ylabel("Score")
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataCor
  y = column_or_1d(y, warn=True)
['x6' 'x12']
accuracies :[135.07762424  36.94342496  13.3129012   35.77163741  12.42958434
  93.73300962 233.92587268  27.57541715  30.27138317 120.66401844
 101.31679539 189.97232058 207.9203739 ]
```



After Feature Selection the data would be as follows:

```
data_train=pd.DataFrame({'flavanoids':our_data[6],'proline':our_data[12]})
data_train
```

	flavanoids	proline
0	3.06	1065.0
1	2.76	1050.0
2	3.24	1185.0
3	3.49	1480.0
4	2.69	735.0
...	...	...
173	0.61	740.0
174	0.75	750.0
175	0.69	835.0
176	0.68	840.0
177	0.76	560.0

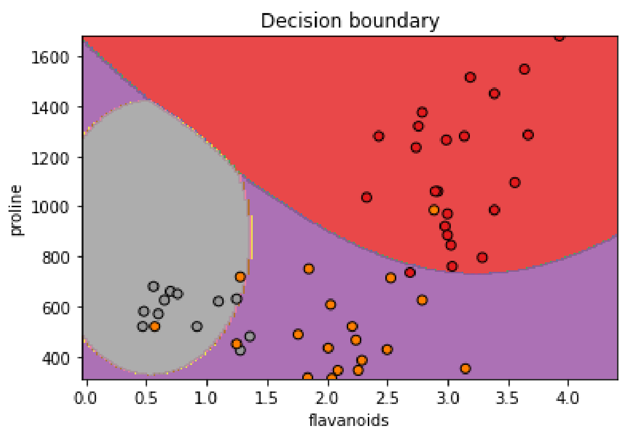
178 rows × 2 columns

c) Plotting the decision boundary on the test set using this function.

```
] # this function can be used to plot the decision boundary
  def plotDecisionBoundary( X, y, model, title=''):
      plt.close('all')
      plt.figure()
      cm = plt.cm.Set1
      x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
      y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
      h = 0.02
      xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                           np.arange(y_min, y_max, h))
      Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
      Z = Z.reshape(xx.shape)
      plt.contourf(xx, yy, Z, cmap=cm, alpha=.8)
      plt.scatter(
          X[:, 0],
          X[:, 1],
          c=y,
          cmap=cm,
          edgecolors='k',
          alpha=1,
      )
      plt.title(title)
```

```
# plot the decision boundary
plotDecisionBoundary(X_TEST.to_numpy(),y_TEST.to_numpy(),model=m,title='Decision boundary')
plt.xlabel('flavanoids')
plt.ylabel('proline')
```

```
Text(0, 0.5, 'proline')
```



(2) KNN classifier:

upload the dataset as data frame from the device.

```
[11] # upload datasets
     file = files.upload()
     #read datasets
     df_original = pd.read_csv('car.csv',header=None)
```

Choose Files car.csv

- **car.csv**(text/csv) - 51867 bytes, last modified: 6/14/2022 - 100% done

```
Saving car.csv to car (1).csv
```

```
print(df_original.head(10))
print(len(df_original))
```

```
       0      1  2  3      4     5      6
0  vhigh  vhigh  2  2  small   low  unacc
1  vhigh  vhigh  2  2  small   med  unacc
2  vhigh  vhigh  2  2  small  high  unacc
3  vhigh  vhigh  2  2    med   low  unacc
4  vhigh  vhigh  2  2    med   med  unacc
5  vhigh  vhigh  2  2    med  high  unacc
6  vhigh  vhigh  2  2    big   low  unacc
7  vhigh  vhigh  2  2    big   med  unacc
8  vhigh  vhigh  2  2    big  high  unacc
9  vhigh  vhigh  2  4  small   low  unacc
1728
```

a) Actually, this step comes after the encoding step in section (b) as we encode all the data first then we shuffle and split it. Here we used the train_test_split function from sklearn.model_selection to randomly select 1300 training and 428 testing data. After that, we separated the 1300 trains into 1000 training and 300 validation data sets

```
df = pd.DataFrame(encoded_data_np)
X_train, X_test, y_train, y_test = train_test_split( df.iloc[:,:-1], df.iloc[:,-1], train_size=1300 ,test_size=428, random_state=42)
X_train, X_val = X_train.iloc[:1000], X_train.iloc[1000:]
y_train, y_val = y_train.iloc[:1000], y_train.iloc[1000:]
df.head(10)
```

	0	1	2	3	4	5	6
0	3	3	0	0	2	1	2
1	3	3	0	0	2	2	2
2	3	3	0	0	2	0	2
3	3	3	0	0	1	1	2

b) The string values need to be converted into ordinal values so we used the OrdinalEncoder function from sklearn. preprocessing to encode all the data values.

```
ordinal_encoder = OrdinalEncoder(dtype = np.uint8)
encoded_data_np = ordinal_encoder.fit_transform(df_original)
encoded_data_np
```

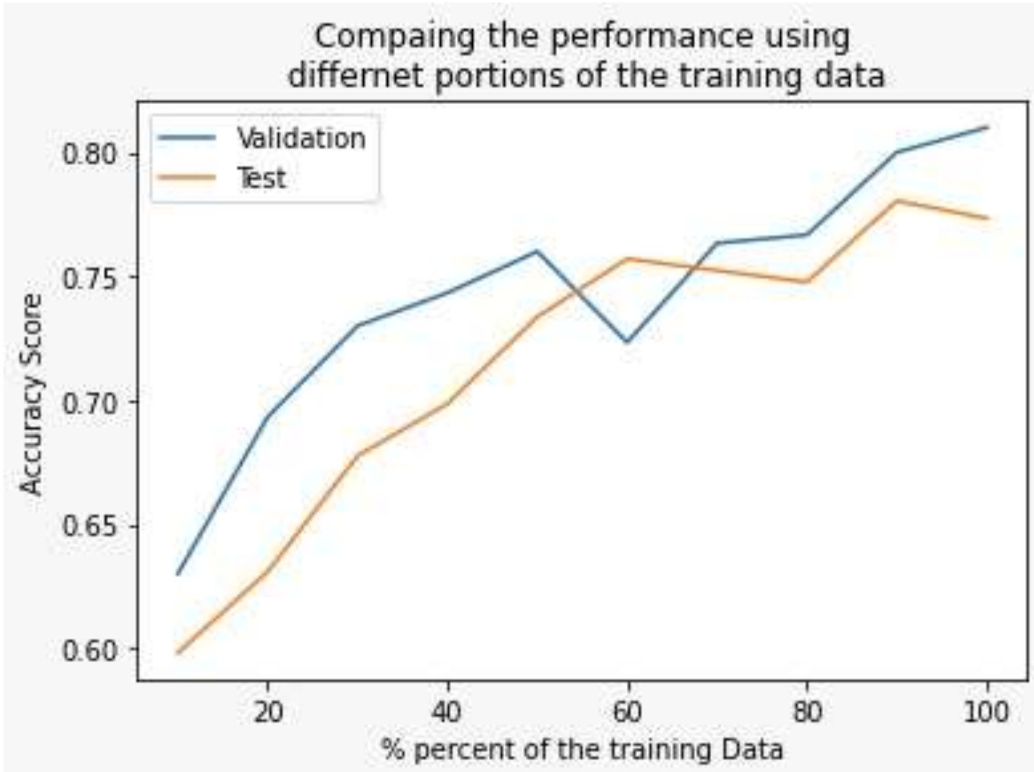
```
array([[3, 3, 0, ..., 2, 1, 2],
       [3, 3, 0, ..., 2, 2, 2],
       [3, 3, 0, ..., 2, 0, 2],
       ...,
       [1, 1, 3, ..., 0, 1, 2],
       [1, 1, 3, ..., 0, 2, 1],
       [1, 1, 3, ..., 0, 0, 3]], dtype=uint8)
```

c) From 100 to 1000 with step size = 100 for each iteration to get the accuracy scores with the validation data and the testing data using 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100% of the training set for 10 separate KNN classifiers with k = 2.

```
KNN_model = KNeighborsClassifier(n_neighbors=2)
x_axes = []
val_scores = []
test_scores = []
for i in range(100,1001,100):
  x_axes.append(i)
  X_train_current = X_train.iloc[:i]
  y_train_current = y_train.iloc[:i]
  KNN_model.fit(X_train_current,y_train_current)
  # get the score for the validation data
  y_val_pred = KNN_model.predict(X_val)
  validation_score = accuracy_score(y_val,y_val_pred)
  val_scores.append(validation_score)

  # get the score for the test data
  y_test_pred = KNN_model.predict(X_test)
  test_score = accuracy_score(y_test, y_test_pred)
  test_scores.append(test_score)
```

Then, we plotted the scores for the validation and testing.

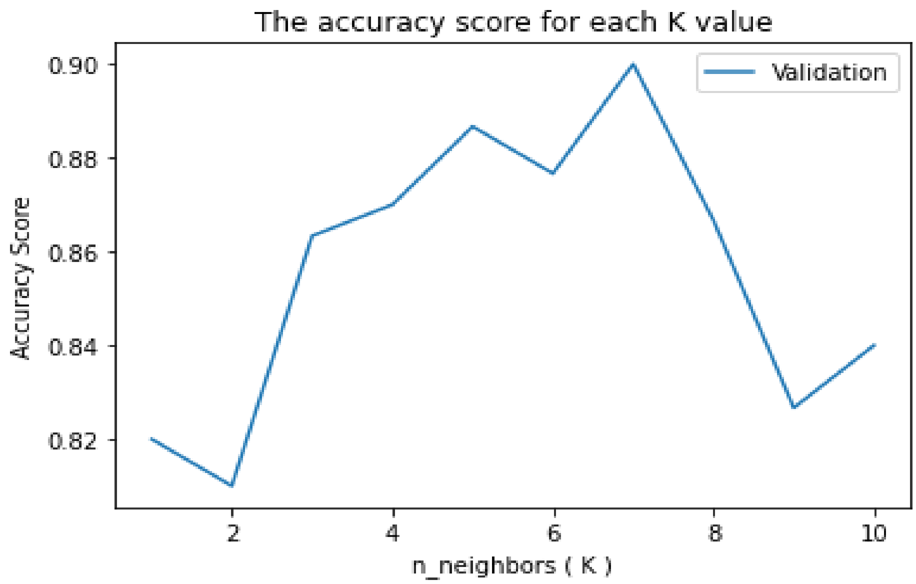


<u>CONCLUSION</u>:

The perfect sweet spot is when the size of the training data is 900 (90%), after that point the validation score is increasing meanwhile the testing score is decreasing which means the model is overfitting.

d) Finding the best k value from 1 to 10 using the accuracy score over the validation set.

```
scores=[]
x_axes =[]
for i in range(1,11,1):
  x_axes.append(i)
  KNN = KNeighborsClassifier(n_neighbors=i)
  KNN.fit(X_train,y_train)
  y_val_pred = KNN.predict(X_val)
  score = accuracy_score(y_val,y_val_pred)
  scores.append(score)
print("scores = ",scores)
```

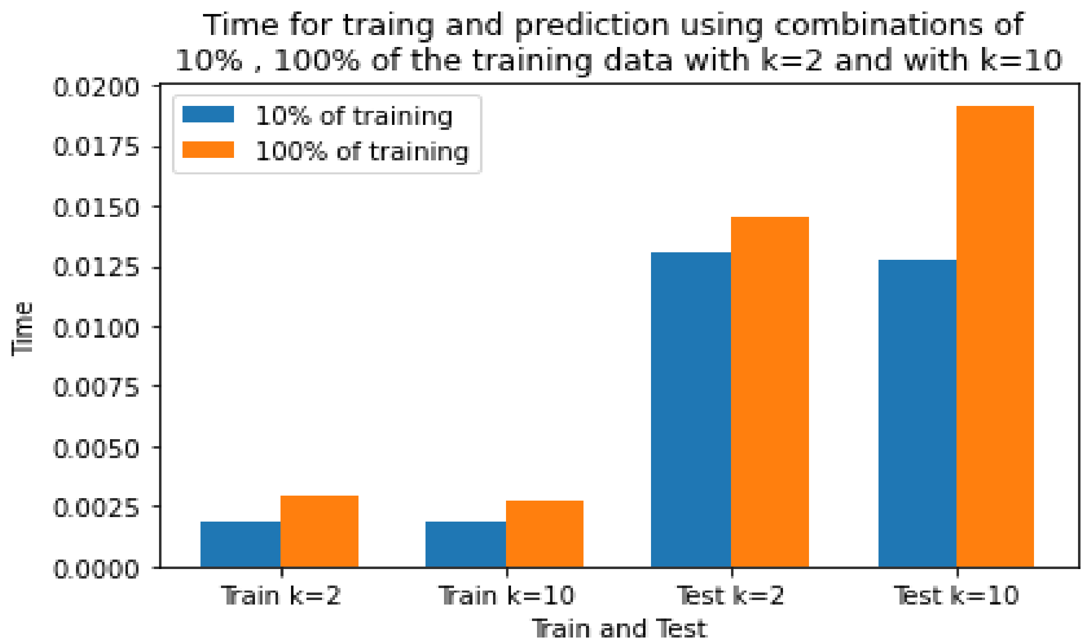


CONCLUSION:

It is obvious now that the best k value is 7 which gives the highest accuracy score.

e) We evaluated the time for training and prediction using the test data.
We had the 4 cases:
- 10% of the whole training set and K = 2
- 100% of the whole training set and K = 2
- 10% of the whole training set and K = 10
- 100% of the whole training set and K = 10.

And the following plot illustrates the results.



<u>CONCLUSION</u>:

In training cases:

The time when k=2 is almost identical to when k=10 and for sure it takes a longer time when we use more data (100%) of the training set as we compute the distances between all the training points.

In testing cases:

When we use 10% of the training set the time when k=2 is almost identical to when k=10 and for sure it takes a longer time when we use more data (100%) as we have more points to compute the distances between them.

The point here is that when we use 100% of training and k=10, it takes more time than when k=2, and that is because we need to compare more points (10) and we may have many classes for these 10 points so need to get the majority of them. Meanwhile, when k=2 it has fewer points to compare with.

In both cases overall:

The prediction for the test data takes a longer time than the training time. During the training phase, the model computes the distances exponentially point by point as we go over the training data but during the testing phase for each test point, we compute the distance between that point and all the trained data to get the majority class of the closest k points.