ELG5255 Applied Machine Learning

Group :19

Assignment No:2

Team members names:

1.Abdulrahman Ahmed

2.Amir Youssef

3.Mohamed Elesawy

Task1: Calculations

Part 1: Using Bayesian Rule Based Classifier to make prediction when Color = G, Gender = F, Price=H . Please include the detailed calculation process.

Color	Target =Y			
	#y	#n	P(y)	P(n)
Green	3	2	3/9	2/6
Total	9	6		

Gender	Target=Y			
	#y	#n	P(y)	P(n)
Female	6	1	6/9	1/6
Total	9	6		

Price	Target=Y			
	#y	#n	P(y)	P(n)
High	2	2	2/9	2/6
Total	9	6		

	#Y=yes	#Y=no	P(Y=yes)	P(Y=no)
Y "Target"	9	6	9/15	6/15
Total	15	15		

A. We will calculate Posterior when Y=Yes

$$P(Y=yes \mid G,F,H)=\frac{P(G|Y)*P(F|Y)*P(H|Y)*P(Y=yes)}{P(G,F,H)}$$

$$\text{Prior } P(Y=yes)=\frac{9}{15}$$

$$P(G,F,H)=P(G,F,H \mid yes)*P(yes)+ P(G,F,H \mid no)*P(no)$$

$$= P(G \mid yes)*P(F \mid yes)*P(H \mid yes)*P(yes)+ P(G \mid no)*P(F \mid no)* P(H \mid no)*P(no)$$

$$P(G,F,H)=3/9*6/9*2/9+9/15+2/6+1/6+2/6+6/15=1/27$$

Posterior P(Y=yes | G,F,H)=$\frac{\frac{3}{9}*\frac{6}{9}*\frac{2}{9}*\frac{9}{15}}{1/27}=\frac{4}{5}=0.8$ ---→1

B. We will calculate Posterior when Y=No

P(Y=no | G,F,H)=$\frac{P(G|Y)*P(F|Y)*P(H|Y)*P(Y)}{P(G,H,F)}$

Prior P(Y=no)=$\frac{6}{15}$

P(G,F,H)=P(G,F,H | yes)*P(yes)+ P(G,F,H | no)*P(no)

= P(G | yes)*P(F | yes)*P(H | yes)*P(yes)+ P(G | no)*P(F | no)* P(H | no)*P(no)

P(G,F,H)=3/9*6/9*2/9+9/15+2/6+1/6+2/6+6/15=1/27

Posterior P(Y=no | G,F,H)=$\frac{\frac{2}{6}*\frac{1}{6}*\frac{2}{6}*\frac{6}{15}}{\frac{1}{27}}=\frac{1}{5}=0.2$ --→2

Posterior (yes | G,F,H)+ Posterior (no | G,F,H)=.2+.8=1

Posterior P(Y=yes | G,F,H) > Posterior P(Y=no | G,F,H)

Part 2: The loss table in question2 in the Calculation part shoule be the following. The second row is the values of choose class2, and the thrid row if the values of choosing class1

Target	Class1	Class2
choose class2	5	2
choose class1	0	5
Reject	4	4

Part Two:

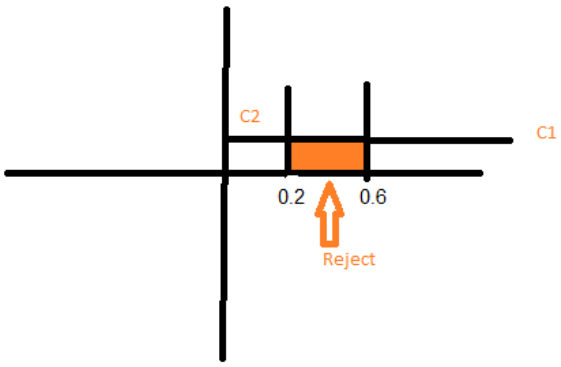
Target	C1	C2
α 2	λ 21=5	λ 22=2
α 1	λ 11=0	λ 12=5
reject	4	4

R*(C1|X) + R*(C2|X) =1 ---> R*(C2|X) =1- R*(C1|X)

R (α1 |X) = λ11*P(C1|X) + λ12*P(C2|X) =0+5*P(C2|X)=5-5R*P(C1|X) -->1
R (α2 |X) = λ21*P(C1|X) + λ22*P(C2|X) =5*P(C1|X) +2*(1-R*P(C1|X))--->2

If we choose α 1	If we choose α 2
R(α 1|X) < Reject 4 5-5R*P(C1|X) < 4 -5R*P(C1|X) <-1 R(C1|X)>1/5	R (α 2|X) < Reject 4 5R*P(C1|X) +2-2R*P(C1|X) < 4 3*P(C1|X) <2 R(C1|X)<2/3

The reject boundary will between: **1/5<**R(C1|X)<2/3



Task2: Programming

Part 1: Use scikitlearn or other python packages to implement a naive Bayesian classifier (GaussianNB), and show the precision, recall, F1 of the testing set. Use wine dataset in the question.

A) The data was split using train_test_split function and a gaussian naïve bayes classifer was trained on the whole data with 13 features:

```
# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 42, stratify = y)

```

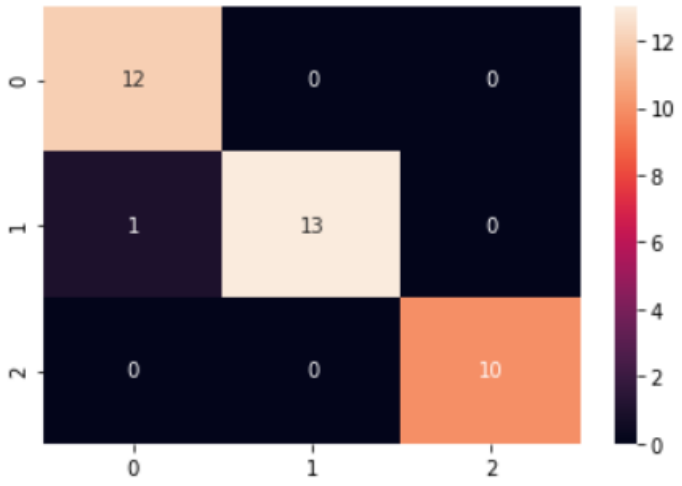
```
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

```
GaussianNB()
```

B) The classification report for the model was printed with accuracy, average precision across classes, average recall across classes, and average f1-score across classes of 97%

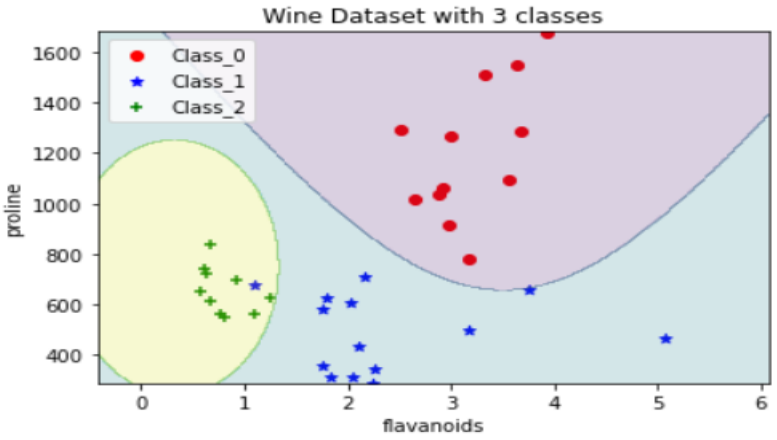
```
print(classification_report(y_test, y_pred, target_names=data.target_names))
```

```
              precision    recall  f1-score   support

     class_0       0.92      1.00      0.96        12
     class_1       1.00      0.93      0.96        14
     class_2       1.00      1.00      1.00        10

    accuracy                           0.97        36
   macro avg       0.97      0.98      0.97        36
weighted avg       0.97      0.97      0.97        36
```



C) The model was then retrained with the two most important features using F-ANOVA test (flavanoids, proline) to plot the decision boundary for the test data.

```
[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 ]
```



Part 2: Use scikit-learn or other python packages to implement a KNN classifier (KNeigh borsClassifier). In this question, we use car-evaluation-dataset:

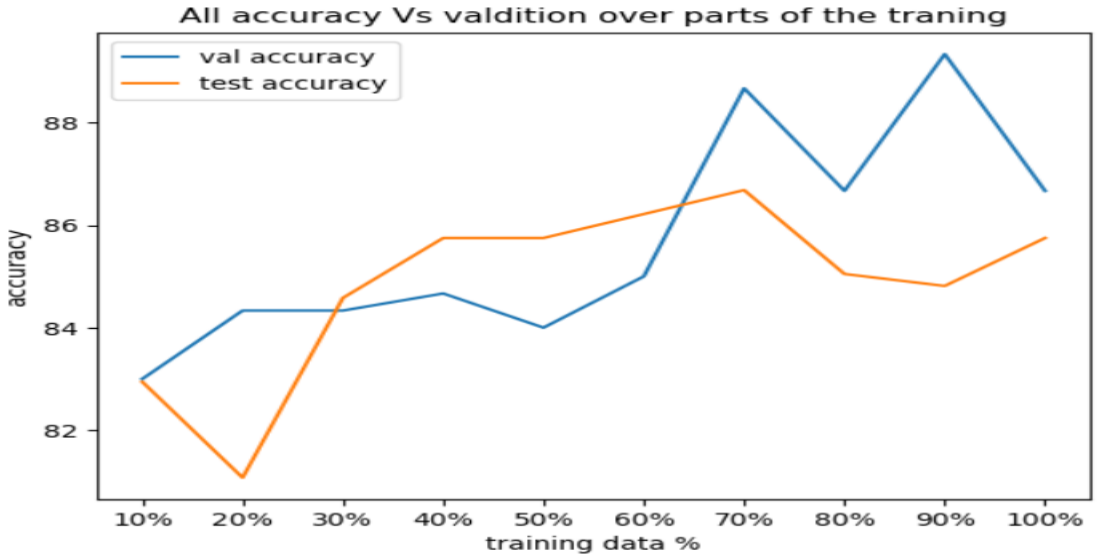
A) The dataset was split into three parts, training set with 1000 samples, validation set with 300 samples, and test set with 428 samples

```
def data_m_split(data):
  data= data.sample(frac=1).reset_index(drop=True)
  X_train=data.iloc[0:1000,:-1]
  y_train=data.iloc[0:1000,-1]
  X_val=data.iloc[1000:1300,:-1]
  y_val=data.iloc[1000:1300,-1]
  X_test=data.iloc[1300:1728,:-1]
  y_test=data.iloc[1300:1728,-1]

  return X_train ,y_train,X_val ,y_val,X_test ,y_test
```

B) All the categorical features in the dataset were converted into numbers using label encoder as a lot of them has ordinal nature

```
def label_the_data(data):
  le = preprocessing.LabelEncoder()

  for i in range(len(data.columns)):
    data.iloc[:,i]=le.fit_transform(data.iloc[:,i])
  return data
```

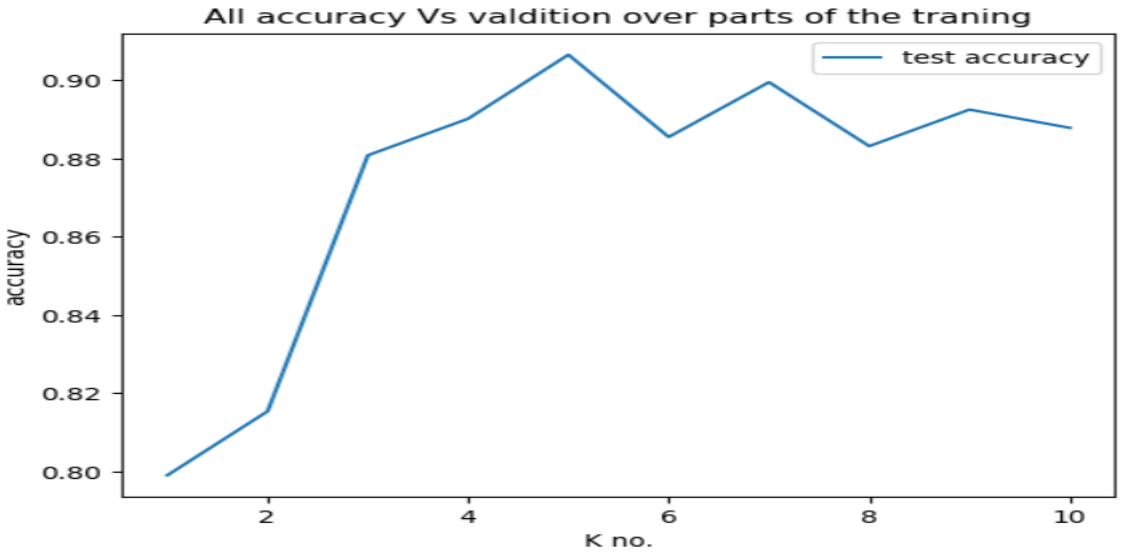
c) 10 models were trained with K=2 each time using an increasing percent of the training set (10%, 20%,30% ….), then the validation and test accuracy of each model was plotted for comparison and analysis of the effect of the portion of training set used on the model accuracy.
(For comments on the graph refer to conclusion section)



And here is an important snippet of code used to train the models:

```
def training_ntimes(X_train ,y_train,X_val ,y_val,X_test ,y_test ,num):
  models=[]
  valllist=[]
  testlist=[]
  for i in range(1,num+1):
    classifier = KNeighborsClassifier(n_neighbors = 2)
    model=classifier.fit(X_train[0:100*i], y_train[0:100*i])
    val=getAccuracy(model, X_val, y_val)

    test=getAccuracy(model, X_test, y_test)
    valllist.append(val)
    testlist.append(test)
  return models,valllist,testlist
```

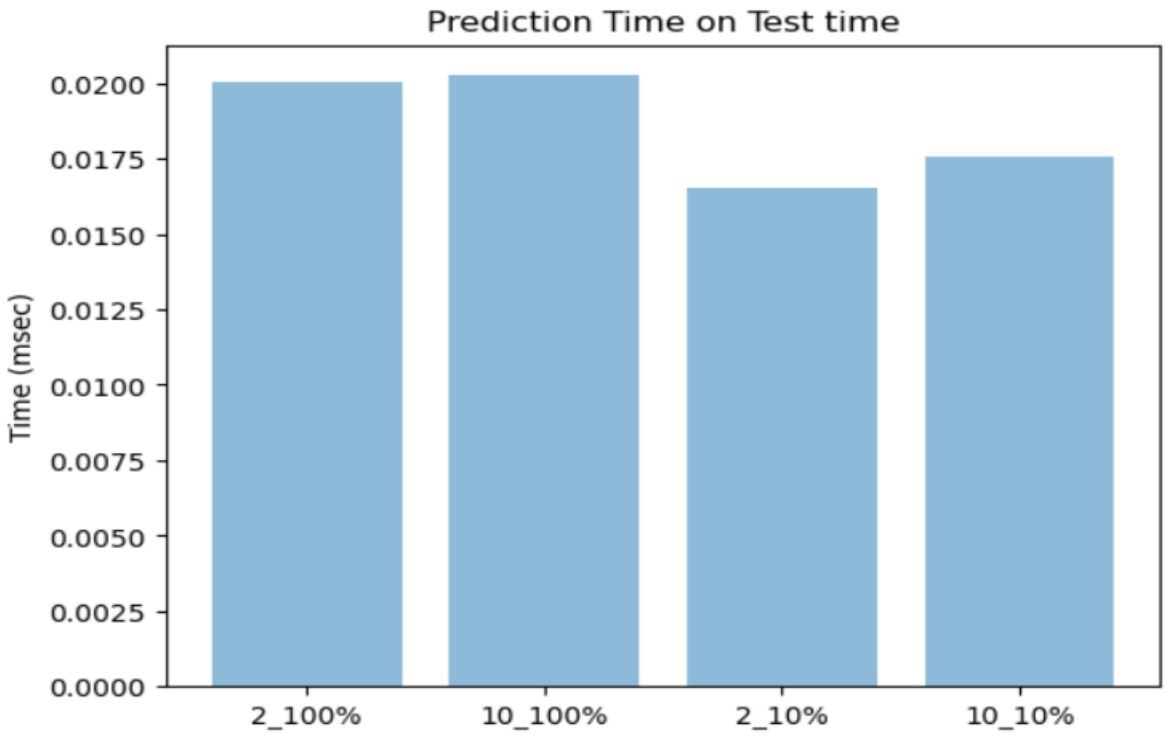
d) 10 model were trained on the whole training dataset with different K values from 1 to 10, the validation accuracy of each model was plotted to figure out the best number for K
(For comments on the graph refer to conclusion section)



And here is an important snippet of code used to train the models:

```
def best_k(X_train ,y_train,X_val ,y_val,X_test ,y_test ,k_range):
# We can create Python dictionary using [] or dict()
  scores = []

  # We use a loop through the range 1 to 26
  # We append the scores in the dictionary
  for k in range(1,k_range+1):
      knn = KNeighborsClassifier(n_neighbors=k)
      knn.fit(X_train, y_train)
      y_pred = knn.predict(X_val)
      scores.append(accuracy_score(y_val, y_pred))

  return scores
```

E) 4 models were trained, one with K=2 and 10% of the training data, one with K=10 and 10% of the training data, one with K=2 and 100% of the training data, and one with K=10 and 100% of the training data, Analyses were made on both training time on training set and prediction time on test set:
(For comments on the graph refer to conclusion section)



Here is an important snippet of code for function timed to produce the above graph:

```
def infcalI(model ,X_y):
  model.predict(X_y)
```

Method and result of timing the training set:

(For comments on the results refer to conclusion section)

```
%timeit -o -n100 -r1  timecalI(X_train ,y_train,2 )
```

```
3.77 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
<TimeitResult : 3.77 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)>
```

```
%timeit -o -n100 -r1   timecalI(X_train ,y_train,10 )

```

```
3.73 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
<TimeitResult : 3.73 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)>
```

```
%timeit -o -n100  -r1  timecalI(X_train[:100] ,y_train[:100],2 )

```

```
1.99 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
<TimeitResult : 1.99 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)>
```

```
%timeit -o -n100 -r1  timecalI(X_train[:100] ,y_train[:100],10 )
```

```
1.96 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
<TimeitResult : 1.96 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)>
```

Here is an important snippet of code for function timed to produce the above result:

```
def timecalI(X_train ,y_train,i):
  classifier = KNeighborsClassifier(n_neighbors = i)
  model=classifier.fit(X_train, y_train)
  return model
```

F) Conclusion:

In part c, although a bit of fluctuation exists, the general trend in the graph describes that both the validation and test accuracy increase by increasing the percentage of the training set used to train the model

In part d, it becomes evident in the graph that the best number for K is 5 as by increasing k more than 5, the validation accuracy decreases

In part e, it appears in the graph of prediction time and through analysis of training time that different numbers of k have nearly the same time while different percentage of training have higher time when increasing the percentage of training set, so k nearly doesn't have an effect on both training and prediction time while amount of data does increase the time taken for both training and prediction.