ELG 5255: Applied Machine Learning

Assignment 2

Group23_HW2

1. Calculations

1- Q1)

NEW INSTANCE ((Color=Green), (Gender=female), (Price=High))

1- Calculate the probability of yes and no
   - P(Y)=9/15=0.6
   - P(N)=6/15=0.4
2- We have 3 features so we will calculate probability of each feature regards to our target.
   - Color

Color	YES	NO
Green	3/9	2/6
Red	2/9	3/6
Yellow	4/9	1/6

   - Gender

Gender	Yes	No
Female	6/9	1/6
Male	3/9	5/6

   - Price

Price	Yes	No
High	2/9	2/6
Medium	2/9	3/6
low	5/9	1/6

3- Calculate new instance
   1- P(New Instance | Y) = P(Y) * P(G|Y) * P(F|Y) * P(H|Y)
      =0.6 * 3/9 * 6/9 * 2/9 = 0.029 = 0.03

   2- P(New Instance | N) = P(N) * P(G|N) * P(F|N) * P(H|N)
      = 0.4* 2/6* 1/6* 2/6 = 1/135

4- P(Y/New instance) = P(New Instance | Y) / P(New/Y) + P(New/n) = 0.80

   P(N/New instance) = P(New Instance | N) / P(New/Y) + P(New/N) = 0.19 = 0.2

The prediction will be yes

2- Q2)

- R(α1|x) = λ11 * P(c1|x) + λ12 * P(c2|x)
  = 0 * P(c1|x) + 5 * P(c2|x)
  1 = P(c1|x) + P(c2|x)
  P(c2|x) = 1 - P(c1|x)
  R(α1|x) = 5(1- P (c1|x))
  5 - 5 * P(c1|x) → 1

- R(α2|x) = λ21 * P(c1|x) + λ22 * P(c2|x)
  =5 * P(c1|x) + 2 * P(c2|x)
  =5 * P((c1|x) + 2 * (1- P (c1|x))
  =3 * P(c1|x) + 2 → 2

- R(α3|x) = λ31 * P(c1|x) + λ32 * P(c2|x)
  =4 * P(c1|x) + 4 * (1- P(c1|x)

From 1 $\boldsymbol{\alpha 1}$: 5-5 * P(c1|x) < 4

5-5 * P(c1|x) < 4

From 1 $\boldsymbol{\alpha 2}$: 3P(c1|x) + 2 < 4

3 * P(c1|x) < 2

From 1 α1	From 1 α2
P(c1|x) > 1/5	P(c1|x) < 2/3

There is no intersection between P(c1|x) < 1/5 and P(c1|x) > 2/3, so no Rejection Area.

2. Programming

We followed some defined steps to obtain the aimed results:

1. (a)

```
# 1(a) ------------------------------------ Read DataSet
DataSet = load_wine()
X = DataSet.data
Y = DataSet.target
```

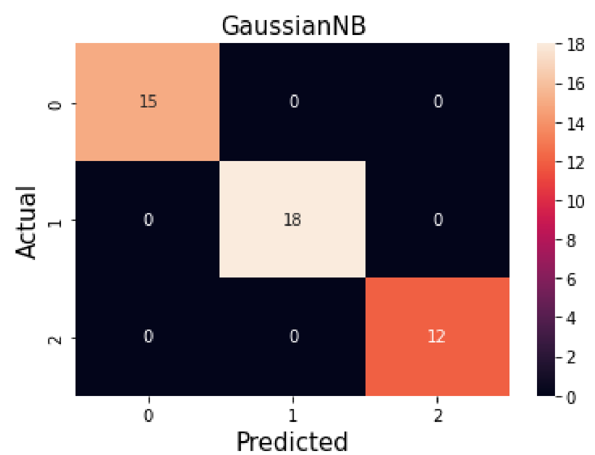
```
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=42)
X_train = X_train.reset_index(drop=True)
X_test = X_test.reset_index(drop=True)
```

1. (b)

Classification report and confusion matrix on GaussianNB() that trained on 13 features.

Acc_OnTest = 100% , Acc_OnTrain = 97.74%

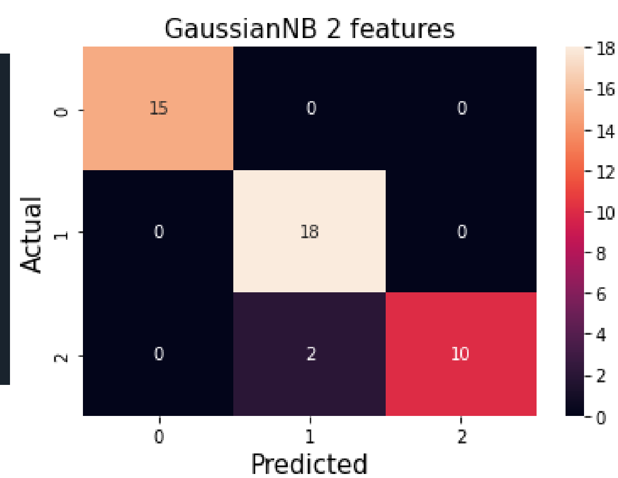
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        18
           2       1.00      1.00      1.00        12

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```



Classification report on GaussianNB() that trained on 2 features **(hue, Proline)** selected by using feature selection method called : **ExtraTreesClassifier()** and **pairplot.**

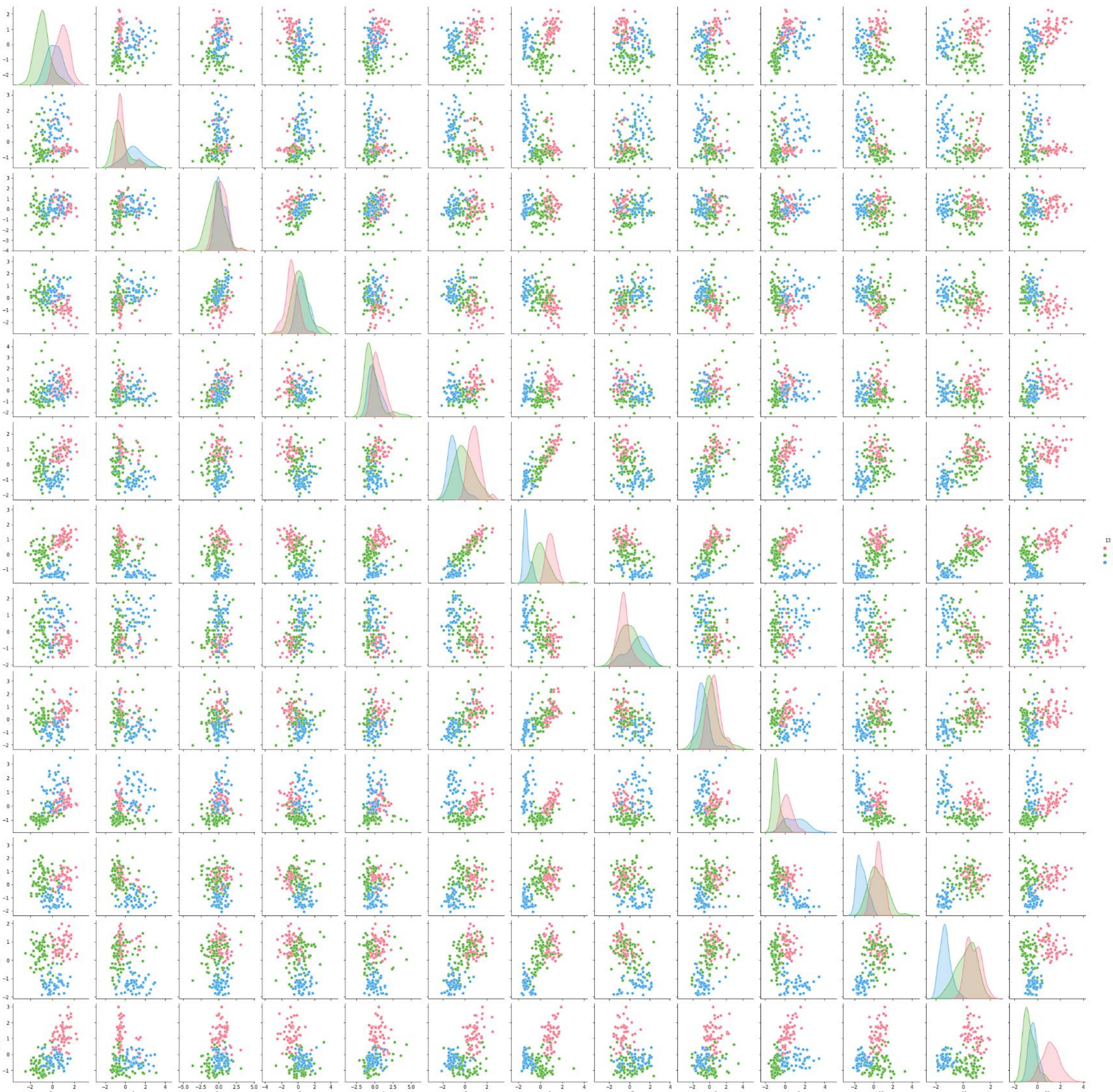
Acc_OnTest = 95.55% , Acc_OnTrain = 88.72%

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       0.90      1.00      0.95        18
           2       1.00      0.83      0.91        12

    accuracy                           0.96        45
   macro avg       0.97      0.94      0.95        45
weighted avg       0.96      0.96      0.95        45
```



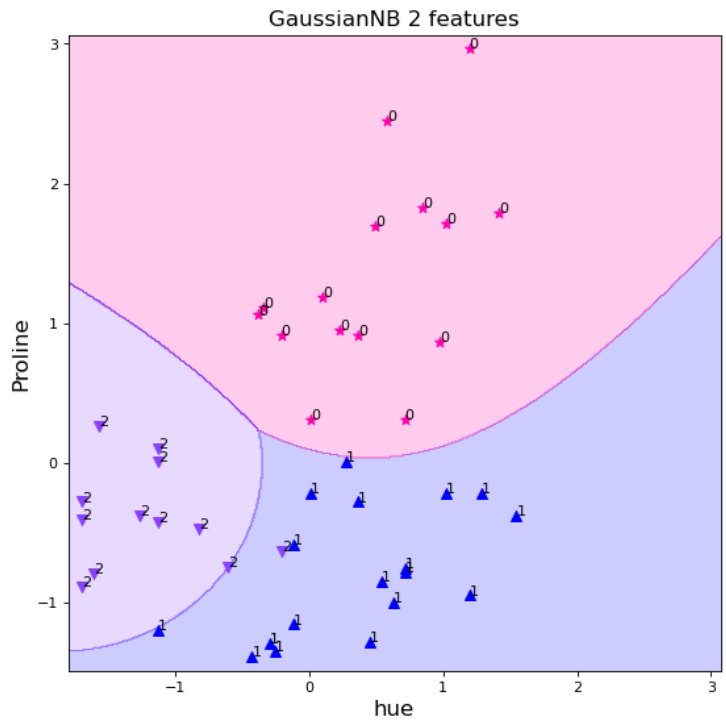
Feature selction function

```
# Function to detemine which features will be eliminated
@staticmethod
def FeatureSelection(X,Y):
    Selector = ExtraTreesClassifier(n_estimators=2)
    Selector = Selector.fit(X, Y)
    return Selector.feature_importances_
```



As displayed above, the model showed lower accuracy when being trained on two features instead of 13. It predicted two false points in the second class as shown in the confusion matrix of model 2. That’s totally make sense, when we train the model on more useful features, it shows higher accuracy and better performance.

1. (c) Plotting the decision boundary.



2. Programming (Part 2)

2. (a)

Reading dataset

```
# 2(a) -------------------------------------
CarDataSet = Assignment2.readDataSet('car_evaluation.csv', 'car_evaluation' , colNames = ['Buying','Maintenance','numOfDoors','numOfPersons','Luggage_boot','Safety','Target'])
```

Shuffle dataset

```
# # 2(a) -------------------------------------- Shuffling the dataset
CarDataSet = CarDataSet.sample(frac=1, random_state = 42).reset_index(drop = True)
X_knn = CarDataSet.iloc[:, 0:6]
Y_knn = CarDataSet.iloc[:, [6]]
```

Split dataset

```
# 2(a) --------------------------------------
# Split Training
X_trainKnn = X_knn.iloc[0:1000, :]
Y_trainKnn = Y_knn.iloc[0:1000, :]

# Split Validation
X_valKnn = X_knn.iloc[1000:1300, :].reset_index(drop=True)
Y_valKnn = Y_knn.iloc[1000:1300, :].reset_index(drop=True)

# Split Test
X_testKnn = X_knn.iloc[1300:, :].reset_index(drop=True)
Y_testKnn = Y_knn.iloc[1300:, :].reset_index(drop=True)
```

2. (b) label encoding

```
# # 2(b) ------------------------------------------- Label encoder
X_knn = Assignment2.labelEncoder(X_knn.copy(),'Buying', ['low','med','high','vhigh'])
X_knn = Assignment2.labelEncoder(X_knn.copy(),'Maintenance', ['low','med','high','vhigh'])
X_knn = Assignment2.labelEncoder(X_knn.copy(),'numOfDoors', ['2','3','4','5more'])
X_knn = Assignment2.labelEncoder(X_knn.copy(),'numOfPersons', ['2','4','more'])
X_knn = Assignment2.labelEncoder(X_knn.copy(),'luggage_boot', ['small','med','big'])
X_knn = Assignment2.labelEncoder(X_knn.copy(),'Safety', ['low','med','high'])

Y_knn = Assignment2.labelEncoder(Y_knn.copy(),'Target', ['unacc','acc','good','vgood'])
```

X_knn - DataFrame

Index	Buying	laintenanc	umOfDoo	mOfPersc	ggage_bo	Safety
0	2	2	2	0	1	2
1	1	0	0	1	1	1
2	2	2	3	0	2	1
3	0	2	3	1	1	1
4	1	0	2	2	1	0
5	1	3	2	1	1	1
6	3	3	0	2	1	2
7	2	0	3	0	2	1
8	1	3	3	2	0	1
9	2	0	0	0	2	2
10	0	2	2	2	1	2
11	1	1	3	1	0	2
12	3	0	0	2	0	2

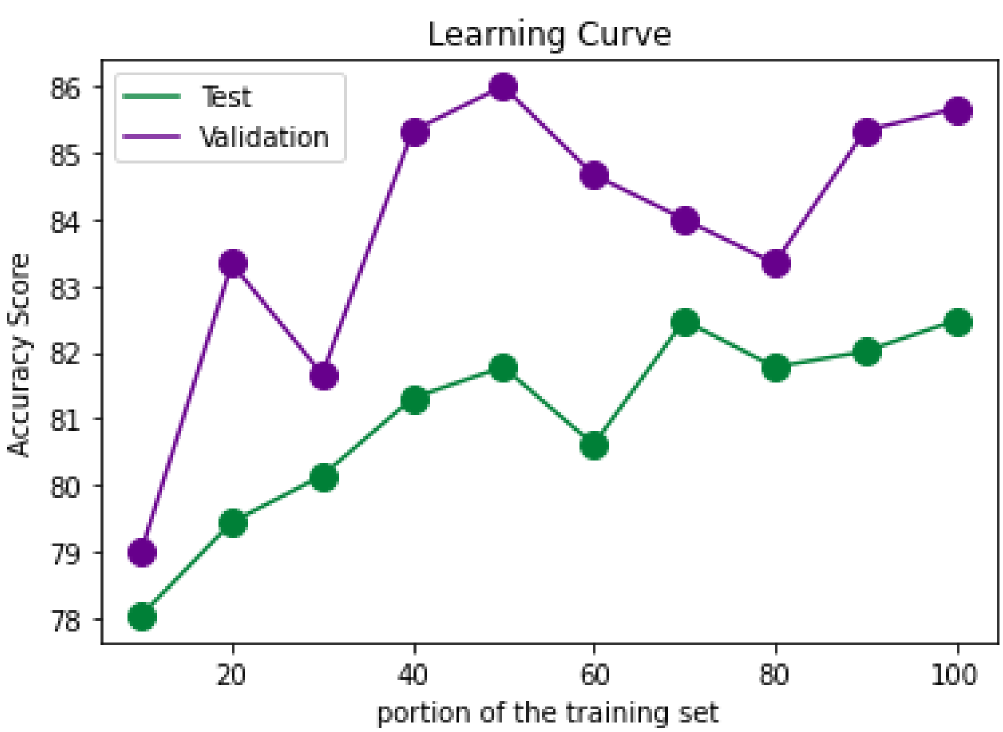
Y_knn - DataFrame

Index	Target
0	0
1	1
2	0
3	1
4	0
5	1
6	0
7	0
8	0
9	0
10	3
11	1
12	0

2. (c)

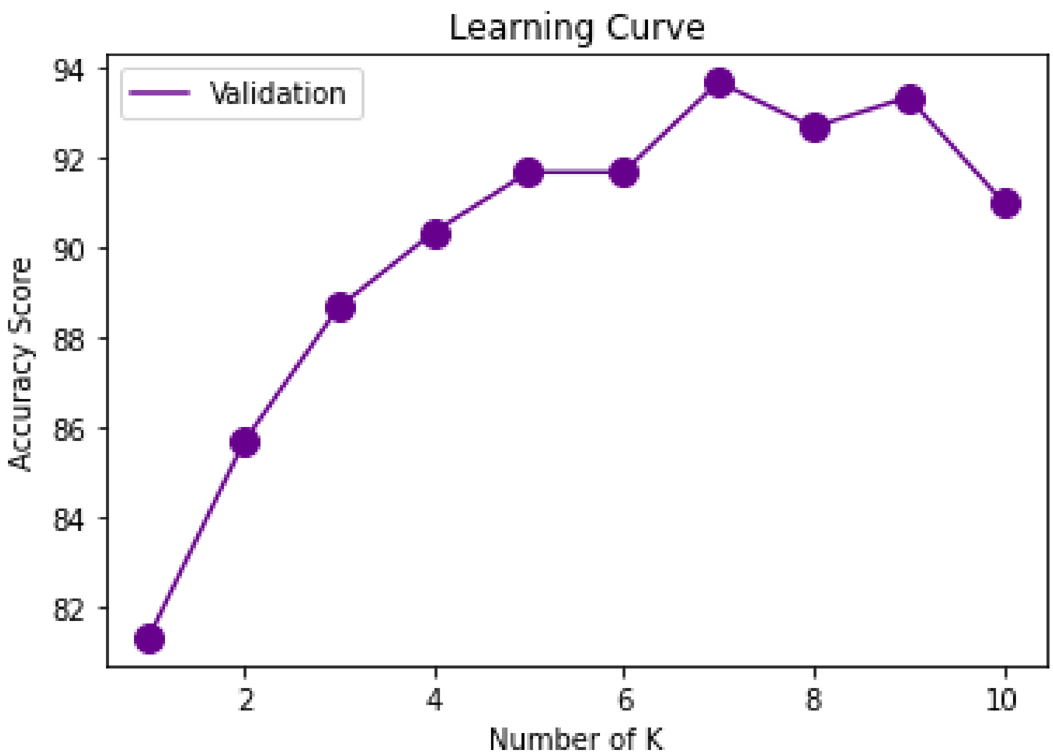
the impact of different number of training samples.

```
 1 - Acc_OnTests (Starting from 10% to 100%) :  78.03738317757009
 1 - Acc_OnVals (Starting from 10% to 100%) :  79.0
*******************************************************************************
 2 - Acc_OnTests (Starting from 10% to 100%) :  79.43925233644859
 2 - Acc_OnVals (Starting from 10% to 100%) :  83.33333333333334
*******************************************************************************
 3 - Acc_OnTests (Starting from 10% to 100%) :  80.14018691588785
 3 - Acc_OnVals (Starting from 10% to 100%) :  81.66666666666667
*******************************************************************************
 4 - Acc_OnTests (Starting from 10% to 100%) :  81.30841121495327
 4 - Acc_OnVals (Starting from 10% to 100%) :  85.33333333333334
*******************************************************************************
 5 - Acc_OnTests (Starting from 10% to 100%) :  81.77570093457945
 5 - Acc_OnVals (Starting from 10% to 100%) :  86.0
*******************************************************************************
 6 - Acc_OnTests (Starting from 10% to 100%) :  80.60747663551402
 6 - Acc_OnVals (Starting from 10% to 100%) :  84.66666666666667
*******************************************************************************
 7 - Acc_OnTests (Starting from 10% to 100%) :  82.4766355140187
 7 - Acc_OnVals (Starting from 10% to 100%) :  84.0
*******************************************************************************
 8 - Acc_OnTests (Starting from 10% to 100%) :  81.77570093457945
 8 - Acc_OnVals (Starting from 10% to 100%) :  83.33333333333334
*******************************************************************************
 9 - Acc_OnTests (Starting from 10% to 100%) :  82.00934579439252
 9 - Acc_OnVals (Starting from 10% to 100%) :  85.33333333333334
*******************************************************************************
 10 - Acc_OnTests (Starting from 10% to 100%) :  82.4766355140187
 10 - Acc_OnVals (Starting from 10% to 100%) :  85.66666666666667
```



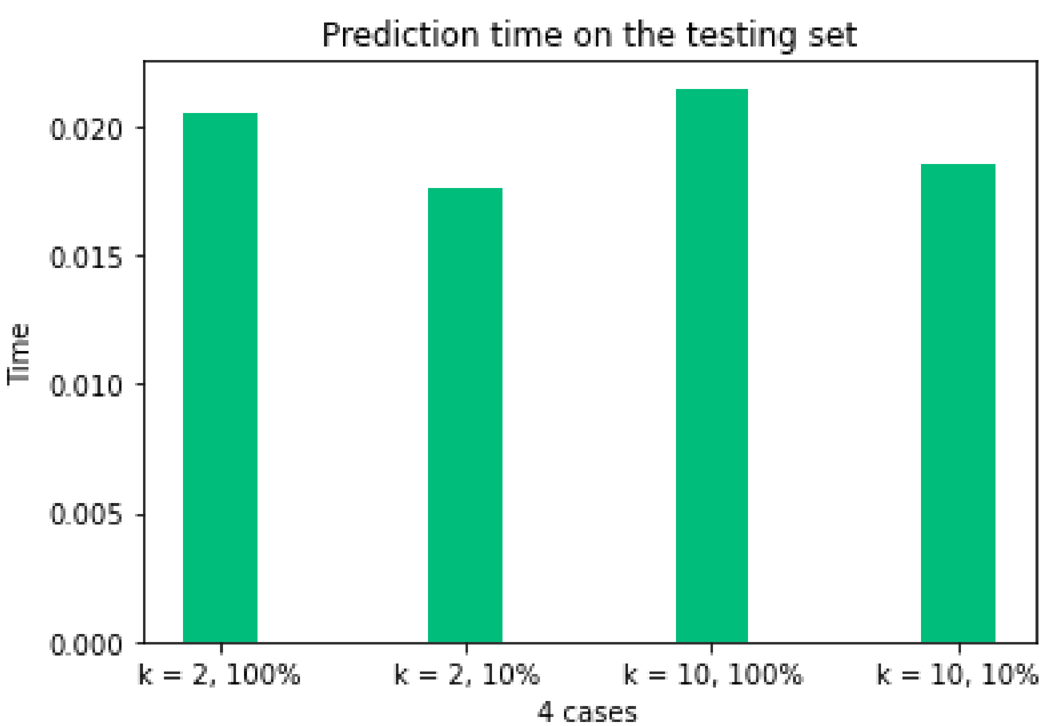
2. (d)

the accuracy curve on the validation set when K varies from 1 to 10.



2. (e)

by using this library **import time,** we have captured the time that each model takes to predict it's results on test set.



time2full	float	1	0.020502328872680664
time2Part	float	1	0.017574310302734375
time10full	float	1	0.02147841453552246
time10Part	float	1	0.018549203872680664

```
classifier_Time_2_full.fit(X_trainKnn.astype('int'),Y_trainKnn["Target"].astype('int'))
classifier_Time_2_part.fit(X_trainKnn_time10.astype('int'),Y_trainKnn_time10["Target"].astype('int'))
classifier_Time_10_full.fit(X_trainKnn.astype('int'),Y_trainKnn["Target"].astype('int'))
classifier_Time_10_part.fit(X_trainKnn_time10.astype('int'),Y_trainKnn_time10["Target"].astype('int'))

start2full = time.time()
y_predsTest_2_full = classifier_Time_2_full.predict(X_testKnn.astype('int'))
stop2full = time.time()
time2full = stop2full - start2full

start2Part = time.time()
y_predsTest_2_part = classifier_Time_2_part.predict(X_testKnn.astype('int'))
stop2Part = time.time()
time2Part = stop2Part - start2Part

start10full = time.time()
y_predsTest_10_full = classifier_Time_10_full.predict(X_testKnn.astype('int'))
stop10full = time.time()
time10full = stop10full - start10full

start10Part = time.time()
y_predsTest_10_part = classifier_Time_10_part.predict(X_testKnn.astype('int'))
stop10Part = time.time()
time10Part = stop10Part - start10Part
```

2. (f) Conclusion

2(c) We have concluded that we should always try different options like playing with different training sizes to obtain different result and choose the best, and based on the graph that we have plotted we notice that when we using (50% to 70%) of training set with KNN we got the best results on validation set and on test set.

2(d) in this point we did something like (hyperparameter tuning) and this process is very useful to determine which combination of parameters will lead to the best results, and based on the graph we have found that when k = 7 we got the best results.

2(e) we have notice that when we increased the sample of training set it takes more time to predict it's results on test set, and also when the k has high value like 10 this sometimes led to more time.