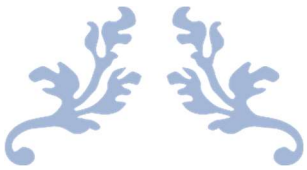
uOttawa



ELG 5255:
Applied Machine Learning
Assignment 2

Dr. Murat Simsek

GROUP 14

Table of Contents

Table of Figures

Part 1:

Question (1):

First calculate the probability of the label to be Yes.

$$P(Target = Y|Color = G, Gender = F, Price = H)$$

$$= \frac{P(Target = Y) * P(Color = G|Target = Y) * P(Gender = F|Target = Y) * P(Price = H|Target = Y)}{P(Color = G, Gender = F, Price = H)}$$

From the given table we can conclude that

$$P(Target = Y) = \frac{9}{15}, P(Color = G|Target = Y) = \frac{3}{9}$$

$$P(Gender = F|Target = Y) = \frac{6}{9}, P(Price = H|Target = Y) = \frac{2}{9}$$

Then

$$P(Target = Y|Color = G, Gender = F, Price = H) = \frac{\frac{9}{15} * \frac{3}{9} * \frac{6}{9} * \frac{2}{9}}{P(Color = G, Gender = F, Price = H)}$$

$$= \boxed{\frac{\frac{4}{135}}{P(Color = G, Gender = F, Price = H)}} \quad \rightarrow \mathbf{1}$$

Second calculate the probability of the label to be No.

$$P(Target = N|Color = G, Gender = F, Price = H)$$

$$= \frac{P(Target = N) * P(Color = G|Target = N) * P(Gender = F|Target = N) * P(Price = H|Target = N)}{P(Color = G, Gender = F, Price = H)}$$

From the given table we can conclude that

$$P(Target = N) = \frac{6}{15}\ , P(Color = \ G|Target = N) = \frac{2}{6}$$

$$P(Gender = \ F|Target = N) = \frac{1}{6}\ , P(Price = H|Target = N) = \frac{2}{6}$$

Then

$$P(Target = N|Color = \ G, Gender = \ F, Price = H) = \frac{\frac{6}{15} * \frac{2}{6} * \frac{1}{6} * \frac{2}{6}}{P(Color = \ G, Gender = \ F, Price = H)}$$

$$= \boxed{\frac{\frac{1}{135}}{P(Color = \ G, Gender = \ F, Price = H)}} \rightarrow \mathbf{2}$$

From 1,2, it's concluded that the prediction will be **Yes** as the probability of Yes is bigger than the probability of No (the denominator is same in both equations).

To calculate the exact values of probabilities, the denominator will be calculated as follows:

$$P(Color = G, Gender = F, Price = H) = \ P(Target = Y) * \ P(Color = \ G|Target = Y)$$

$$* \ P(Gender = \ F|Target = Y) * \ P(Price = H|Target = Y) + \ P(Target = N)$$

$$* \ P(Color = \ G|Target = N) * P(Gender = \ F|Target = N)$$

$$* P(Price = H|Target = N) = \frac{4}{135} + \frac{1}{135} = \frac{5}{135}$$

Then

$$P(Target = Y|Color = G, Gender = F, Price = H) = \frac{\frac{4}{135}}{P(Color = G, Gender = F, Price = H)}$$

$$= \frac{\frac{4}{135}}{\frac{5}{135}} = \mathbf{0.8}$$

$$P(Target = N|Color = G, Gender = F, Price = H) = \frac{\frac{1}{135}}{P(Color = G, Gender = F, Price = H)}$$

$$= \frac{\frac{1}{135}}{\frac{5}{135}} = \mathbf{0.2}$$

Question (2):

The expected risk of three actions will be as follows:

$$R(\alpha_1|x) = \sum_{k=1}^{K} \lambda_{1k} * P(C_k|x) = 0 * P(C_1|x) + 5 * P(C_2|x) = 5 * \big(1 - P(C_1|x)\big)$$

$$= 5 - 5 * P(C_1|x)$$

$$R(\alpha_2|x) = \sum_{k=1}^{K} \lambda_{2k} * P(C_k|x) = 5 * P(C_1|x) + 2 * P(C_2|x)$$

$$= 5 * P(C_1|x) + 2 * \big(1 - P(C_1|x)\big) = 3 * P(C_1|x) + 2$$

$$R(\alpha_r|x) = 4$$

Choose α_1 if

$R(\alpha_1|x) < R(\alpha_r|x)$ and $R(\alpha_1|x) < R(\alpha_2|x)$

$5 - 5 * P(C_1|x) < 4$ $\quad$ $5 - 5 * P(C_1|x) < 3 * P(C_1|x) + 2$

$5 - 4 < 5 * P(C_1|x)$ $\quad$ $5 - 2 < 8 * P(C_1|x)$

$\boxed{P(C_1|x) > \frac{1}{5} \rightarrow \mathbf{1}}$ $\quad$ $\boxed{P(C_1|x) > \frac{3}{8} \rightarrow \mathbf{2}}$

Then from 1,2, it's concluded that the action α_1 will be taken if $P(C_1|x) > \frac{3}{8}$ **→ 3**

Choose α_2 if

$R(\alpha_2|x) < R(\alpha_r|x)$ and $R(\alpha_1|x) > R(\alpha_2|x)$

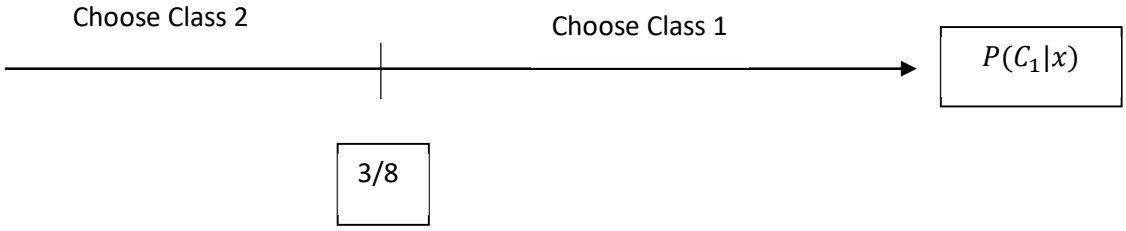
$3 * P(C_1|x) + 2 < 4$ $\quad$ $5 - 5 * P(C_1|x) > 3 * P(C_1|x) + 2$

$3 * P(C_1|x) < 4 - 2$ $\quad$ $5 - 2 > 8 * P(C_1|x)$

$\boxed{P(C_1|x) < \frac{2}{3} \rightarrow \mathbf{4}}$ $\quad$ $\boxed{P(C_1|x) < \frac{3}{8} \rightarrow \mathbf{5}}$

Then from 4,5, it's concluded that the action α_2 will be taken if $P(C_1|x) < \frac{3}{8}$ **→ 6**

Then from 3,6, **<u>the rejection area will be none</u>** and the boundaries between areas will be as follows:



Part 2

1.

(A)

Using train_test_split function from Scikitlearn.

```
[7]  1 x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.20,random_state=42,stratify=y )
```

```
[8]  1 print( y_train)
```

```
[0 0 0 0 2 2 1 2 0 0 1 1 0 0 2 1 1 1 0 0 1 2 0 2 2 0 0 1 1 0 0 0 0 0 1 2 0
 1 0 1 1 1 1 1 1 1 0 0 2 0 0 2 1 1 2 1 1 1 1 0 1 1 0 0 0 0 0 1 1 1 2 2 0 0
 2 1 1 1 1 2 0 2 0 2 1 2 2 1 0 0 1 1 1 0 0 0 2 1 1 1 2 1 0 1 0 1 2 0 0 1 2
 2 1 1 1 2 2 2 2 1 1 2 1 2 2 0 0 2 0 1 2 2 1 2 1 1 1 2 1 0 2 2]
```

Figure 1 train_test_split

(B)

Classification report to calculate precision, recall and F1 for test data.

```
10]  1 y_pred=Parametric_model.predict(x_test)
     2 print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.92	1.00	0.96	12
1	1.00	0.93	0.96	14
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

Figure 2 Classification Report for Test data

(C)

In Plotting the decision boundary of the test dataset, ANOVA was used to select the two features, the selected features are ("flavanoids" and "proline") and then the model is retrained using these two features and finally the decision boundary is plotted.

```
1 plotDecisionBoundary(x_test,y_test,model,"naive Bayesian classifier (GaussianNB ) Model","flavanoids","proline")
```

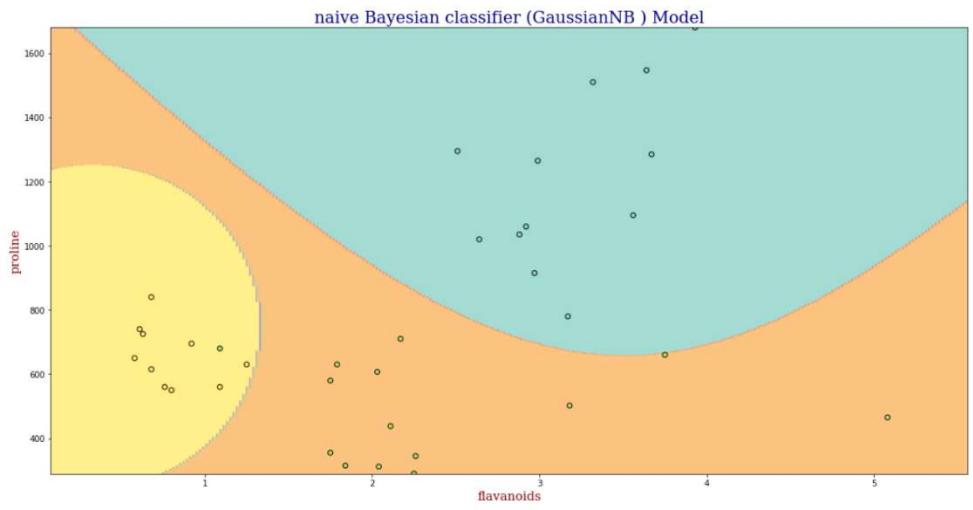


Figure 3 Decision Boundary for test dataset

2.

(A)

Shuffling the dataset and splitting the dataset into a training set with 1000 samples and a validation set with 300 samples and a testing set with 428 samples.

```
]  1 print(car_df.iloc[:,:].shape)
   2 X=car_df.iloc[:,:-1]
   3 y=car_df.iloc[:,-1]
   4 x_train, x_test_valid, y_train, y_test_valid = train_test_split(X, y, train_size=1000,random_state=42,stratify=y )
   5 x_test, x_valid, y_test, y_valid = train_test_split(x_test_valid, y_test_valid, test_size=300,random_state=42,stratify=y_test_valid )
   6
   7 print(x_train.shape)
   8 print(x_test.shape)
   9 print(x_valid.shape)
```

```
(1728, 7)
(1000, 6)
(428, 6)
(300, 6)
```

Figure 4 Shuffling and splitting the dataset

(B)

Distance metric - transforming the string values into numbers.

```
1 def mapping(x):
2     mapper_Price_MC={'low':1,'med':2,'high':3,'vhigh':4}
3     mapper_safety={'low':1,'med':2,'high':3}
4     mapper_lug_boot={'small':1,'med':2,'big':3}
5     mapper_nPerson={'2':2,'3':3,'4':4,'more':5}
6     mapper_nDoor={'2':2,'3':3,'4':4,'5more':5}
7     x.loc[:,('buying price')]=x.loc[:,('buying price')].replace(mapper_Price_MC)
8     x.loc[:,('maintenance cost')]=x.loc[:,('maintenance cost')].replace(mapper_Price_MC)
9     x.loc[:,('number of doors')]=x.loc[:,('number of doors')].replace(mapper_nDoor)
10    x.loc[:,('number of persons')]=x.loc[:,('number of persons')].replace(mapper_nPerson)
11    x.loc[:,('lug_boot')]=x.loc[:,('lug_boot')].replace(mapper_lug_boot)
12    x.loc[:,('safety')]=x.loc[:,('safety')].replace(mapper_safety)
13    return x
```

```
1 x_train=mapping(x_train)
2 x_test=mapping(x_test)
3 x_valid=mapping(x_valid)
4 print(x_train)
5
```

```
      buying price  maintenance cost  number of doors  number of persons  \
913              2                 4                3                  5
1011             2                 3                3                  4
399              4                 1                4                  5
962              2                 4                5                  4
91               4                 4                5                  4
...            ...               ...              ...                ...
15               4                 4                2                  4
1352             1                 4                4                  2
1278             2                 1                5                  4
842              3                 1                5                  2
1460             1                 3                4                  2

      lug_boot  safety
913          2       2
1011         2       1
399          2       1
962          3       3
91           1       2
...        ...     ...
15           3       1
1352         1       3
1278         1       1
842          2       3
1460         1       3

[1000 rows x 6 columns]
```

Figure 5 Transforming strings into numeric values

(C)

Train and test non-Parametric model (KNN) on multi size of data and note the relation between data size and accuracy.

```
size=x_train.shape[0]
train_size=[]
test_accuracy=[]
valid_accuracy=[]
for i in range(1,11):
    non_Parametric_model = KNeighborsClassifier(n_neighbors=2)
    end=int(i*0.1*size)
    non_Parametric_model.fit(x_train.iloc[:end,:], y_train.iloc[:end])

    train_size.append(str(i*10) +'%' )

    y_pred_v=non_Parametric_model.predict(x_valid)
    valid_accuracy.append(accuracy_score(y_valid, y_pred_v))

    y_pred_t=non_Parametric_model.predict(x_test)
    test_accuracy.append(accuracy_score(y_test, y_pred_t))
plt.figure(figsize=(20,10))
print(test_accuracy)
print(valid_accuracy)
plt.plot(train_size, test_accuracy, label = "Test Accuracy")
plt.plot(train_size, valid_accuracy, label = "Validation Accuracy")

font1 = {'family':'serif','color':'darkblue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.title("The effect the change of size of train data on KNN model ", fontdict = font1)
plt.xlabel("Size of Train Data (Percentage)", fontdict = font2)
plt.ylabel("Accuracy", fontdict = font2)
plt.legend()
plt.show()
```

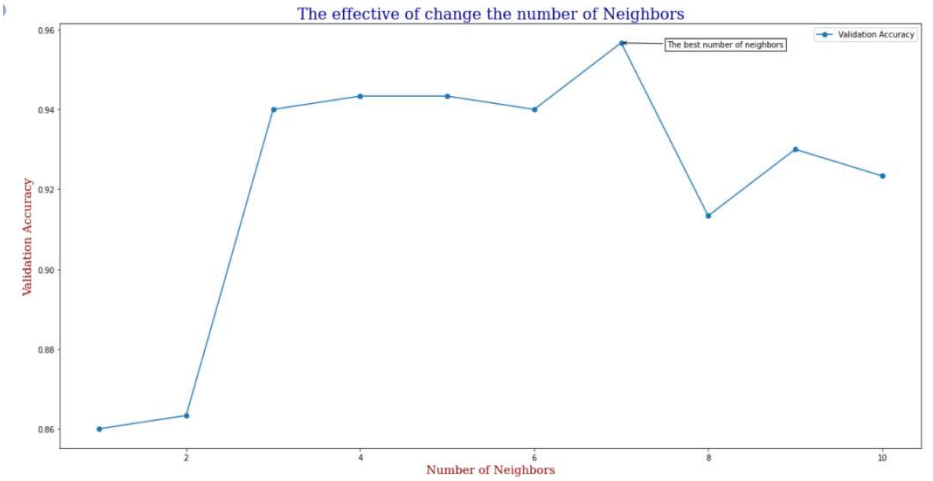
```
[0.780373831775701, 0.8130841121495327, 0.8317757009345794, 0.8621495327102804, 0.8504672897196262, 0.8457943925233645, 0.8504672897196262, 0.852803738317757, 0.8481308411214953, 0.8621495327102804]
[0.7466666666666667, 0.8333333333333334, 0.8466666666666667, 0.8366666666666667, 0.8533333333333334, 0.8633333333333333, 0.87, 0.8533333333333334, 0.87, 0.8633333333333333]
```

Figure 6 Train and test non-Parametric model (KNN) on multi size of data

(D)

Train and test non-Parametric model (KNN) on multi number of neighbors and get the best number of neighbors it reaches highest accuracy

```
train_size=[]
valid_accuracy=[]
for i in range(1,11):
    non_Parametric_model = KNeighborsClassifier(n_neighbors=i)
    non_Parametric_model.fit(x_train, y_train)

    train_size.append(i )

    y_pred_v=non_Parametric_model.predict(x_valid)
    valid_accuracy.append(accuracy_score(y_valid, y_pred_v))

fig =plt.figure(figsize=(20,10))
ax = fig.add_subplot()
ax.plot(train_size, valid_accuracy, label = "Validation Accuracy",marker='o')


plt.title("The effective of change the number of Neighbors", fontdict = font1)
plt.xlabel("Number of Neighbors", fontdict = font2)
plt.ylabel("Validation Accuracy", fontdict = font2)
max_valid=max(valid_accuracy)
maxInd=np.argmax(np.array(valid_accuracy))
best_N_Neighbor=train_size[maxInd]
bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k")
arrowprops=dict(arrowstyle="->")
kw = dict(xycoords='data',textcoords="axes fraction",
          arrowprops=arrowprops, bbox=bbox_props, ha="right", va="top")
ax.annotate("The best number of neighbors", xy=(best_N_Neighbor, max_valid), xytext=(0.84,0.96), **kw)

plt.legend()
plt.show()
print(f'\n The best number of neighbors is : {best_N_Neighbor}')
```




```
The best number of neighbors is : 7
```

Figure 7 Train and test non-Parametric model (KNN) on multi number of neighbours

(E)

Analyzing the training time and testing time when using different number of training samples and different number of neighbors. At 10% of the whole training set and K = 2, while at 100% of the whole training set and K = 2. At 10% of the whole training set and K = 10, while at 100% of the whole training set and K = 10.

```
train_time=[]
test_time=[]
d=[]
# • 10% of the whole training set and K = 2
KNN = KNeighborsClassifier(n_neighbors=2)
start = timeit.default_timer()
KNN.fit(x_train.iloc[:100,:], y_train.iloc[:100])
end = timeit.default_timer()
train_time.append(end - start)
d.append(end - start)
start = timeit.default_timer()
y_pred_t=KNN.predict(x_test)
end = timeit.default_timer()
test_accuracy1=accuracy_score(y_test, y_pred_t)
test_time.append(end - start)
d.append(end - start)
#• 100% of the whole training set and K = 2
KNN = KNeighborsClassifier(n_neighbors=2)
start = timeit.default_timer()
KNN.fit(x_train, y_train)
end = timeit.default_timer()
train_time.append(end - start)
d.append(end - start)
start = timeit.default_timer()
y_pred_t=KNN.predict(x_test)
end = timeit.default_timer()
test_accuracy1=accuracy_score(y_test, y_pred_t)
test_time.append(end - start)
d.append(end - start)
#• 10% of the whole training set and K = 10
KNN = KNeighborsClassifier(n_neighbors=10)
start = timeit.default_timer()
KNN.fit(x_train.iloc[:100,:], y_train.iloc[:100])
end = timeit.default_timer()
train_time.append(end - start)
d.append(end - start)
start = timeit.default_timer()
y_pred_t=KNN.predict(x_test)
end = timeit.default_timer()
test_accuracy1=accuracy_score(y_test, y_pred_t)
test_time.append(end - start)
d.append(end - start)
#• 100% of the whole training set and K = 10
KNN = KNeighborsClassifier(n_neighbors=10)
start = timeit.default_timer()
KNN.fit(x_train, y_train)
end = timeit.default_timer()
train_time.append(end - start)
d.append(end - start)
start = timeit.default_timer()
```

```
y_pred_t=KNN.predict(x_test)
end = timeit.default_timer()
test_accuracy1=accuracy_score(y_test, y_pred_t)
test_time.append(end - start)
d.append(end - start)
print(train_time)
print(test_time)
plt.figure(figsize=(20,10))
g = sns.barplot(x=['•10% training set & K = 2','•10% training set & K = 2','•100% trainin
g set & K = 2','•100% training set & K = 2','•10% training set & K = 10','•10% training s
et & K = 10','•100% training set & K = 10','•100% training set & K = 10'], y=d,hue=['Trai
n Time','Test Time','Train Time','Test Time','Train Time','Test Time','Train Time','Test
Time'])
font1 = {'family':'serif','color':'darkblue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.title("The effect the change of size of train data on KNN model ", fontdict = font1)
plt.xlabel("Models", fontdict = font2)
plt.ylabel("Time (second)", fontdict = font2)
plt.legend()
plt.show()
```

```
[0.007783458999711001 5, 0.003847369000141043, 0.0026032900000245718, 0.003741299000012077]
[0.019215368999994098, 0.022227707999718405, 0.021803527999963 99, 0.025679498000044987]
```

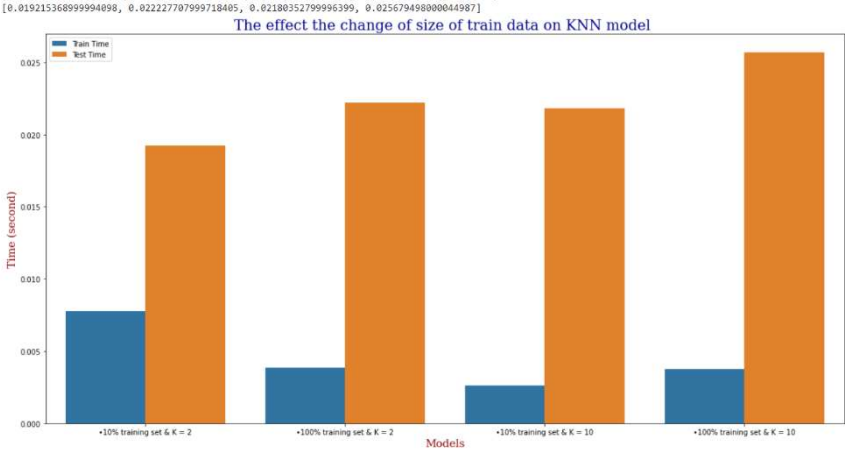


Figure 8 The effect the change of size of train data on KNN model

(F)

In part (C), from the figure, it's concluded that increasing the size of training data will increase test and validation accuracies. In part (D), from the figure, it's concluded that the best k value is 7 (it achieved the highest accuracy). In part (E), from the figure, it's concluded that the prediction time is larger than the training time (because the KNN algorithm is a lazy learner as it calculates the Euclidean distance between the test point and all the training data to get the nearest K neighbors and detect which class based on the majority of k neighbors and all these calculations happen in prediction. Changing the number of k neighbors will affect a little bit the prediction time because the model already calculated the Euclidean distance between the prediction point and all training data. When the train data size is increased the prediction time will increase because the KNN is a nonparametric algorithm and it uses train data in the prediction.