Classification : Nearest Neighbors and Naive Bayes

Assignment 5a | CH17B066

Classification using Nearest Neighbors

(a) Perform k-Nearest neighbours on the given dataset(X_{knn} and y_{knn}: where X_{knn} stores feature vectors representing the movies and y_{knn} stores the 0-1 labelling for each movie) for binary classification of movies, for classifiying whether a given movie is a comedy(label 1) or not a comedy(label 0) . Split the dataset into train(80%), validation(10%) and test sets(10%).Run k-Nearest neighbours for different k values (1,3,7,15,31,63). Select the k, using validation set, which returns the best accuracy score.

(i) Report all the validation accuracies for all the values of k.
(ii) Report accuracy score by performing k-NN on the test dataset using the best chosen k value.

In [1]:

```
#Loading libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split
```

In [2]:

```
#Loading the data
#xknn = pd.read_csv('X_knn.csv', delimiter=" ", header=None)
#yknn = pd.read_csv('y_knn.csv', header=None)

xknn = np.loadtxt('X_knn.csv', delimiter=" ")
yknn = np.loadtxt('y_knn.csv')
```

In [3]:

```
yknn.shape
```

Out[3]:

```
(10000,)
```

In [4]:

```
#Splitting data into training, validation and testing sets
X_train, X_val, y_train, y_val = train_test_split(xknn, yknn, test_size=0.2, random_sta
te=42)

X_val, X_test, y_val, y_test = train_test_split(X_val, y_val, test_size=0.5, random_sta
te=42)

print(y_train.shape, y_val.shape, y_test.shape)
```

```
(8000,) (1000,) (1000,)
```

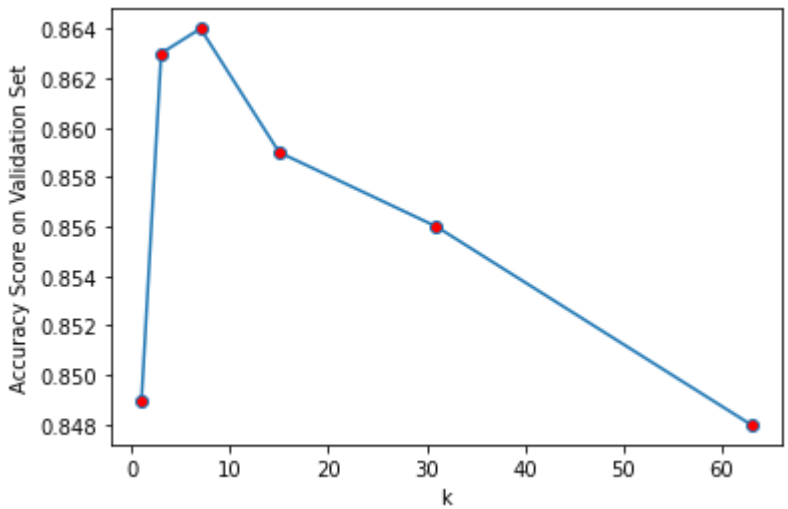
In [5]:

```
#Applying K-NN
from sklearn.neighbors import KNeighborsClassifier

#k values: (1,3,7,15,31,63)
k_vals = [1,3,7,15,31,63]
scores=[]
for k in k_vals:
    Model = KNeighborsClassifier(n_neighbors = k)
    Model.fit(X_train,y_train)
    score = Model.score(X_val,y_val)
    scores.append(score)
```

In [6]:

```
#Plotting for different k:
plt.plot(k_vals,scores, linestyle='-', marker='o', markerfacecolor ='r');
plt.xlabel('k')
plt.ylabel('Accuracy Score on Validation Set')

bestK = k_vals[scores.index(max(scores))]
print('Value of k which performs best on validation set is',bestK)
```

```
Value of k which performs best on validation set is 7
```



In [7]:

```
#Test accuracy for k=7
Model = KNeighborsClassifier(n_neighbors = 7)
Model.fit(X_train,y_train)
test_score = Model.score(X_test,y_test)
print("Test accuracy for k=7 is",test_score)
```

```
Test accuracy for k=7 is 0.863
```

Write the results here

In [8]:

```
print("Validation accuracy for each k is")
for i in range(len(k_vals)):
    print("k = %d: Validation Accuracy = %.3f" %(k_vals[i],scores[i]))
    i+=1
print('\nValue of k which performs best on validation set is',bestK)
print("Test accuracy for k=7 is",test_score)
```

```
Validation accuracy for each k is
k = 1: Validation Accuracy = 0.849
k = 3: Validation Accuracy = 0.863
k = 7: Validation Accuracy = 0.864
k = 15: Validation Accuracy = 0.859
k = 31: Validation Accuracy = 0.856
k = 63: Validation Accuracy = 0.848

Value of k which performs best on validation set is 7
Test accuracy for k=7 is 0.863
```

(b) State why using an even value of k in k-NN should not be chosen

Even values for k are not chosen mainly to avoid any ties with class label scores (distance) during classification.

Learning Naive Bayes' classifier

From Continuous Distribution of data

Here, the distribution of the data(X represents the datapoints and Y represents the 0-1 binary-class label; where 0 being the negative class and 1 being the positive class) is already known.
Consider the following one-dimensional(1-D) Gaussian distributions where means and variances are unknown. You need to estimate means(μ_- : for negative class and μ_+ : for positive class) and variances (σ_-^2 : for negative class and σ_+^2: for positive class) from the given data :
(1) Assume $X|Y_{Y=0} \sim \mathcal{N}(\mu_-, \sigma_-^2)$
(2) Assume $X|Y_{Y=1} \sim \mathcal{N}(\mu_+, \sigma_+^2)$

Generating artificial datasets in the next cell

In [9]:

```
## This cell is for generating datasets. Students should not change anything in this ce
ll.
## You can compare your mean and variance estimates by the actual ones used to generate
these datasets

import numpy as np
X_pos = np.random.randn(1000,1)+np.array([[2.]])
X_neg = np.random.randn(1000,1)+np.array([[4.]])
X_train_pos = X_pos[:900]
X_train_neg = X_neg[:900]
X_test_pos = X_pos[900:]
X_test_neg = X_neg[900:]
X_train = np.concatenate((X_train_pos, X_train_neg), axis=0)
X_test = np.concatenate((X_test_pos, X_test_neg), axis=0)
Y_train = np.concatenate(( np.ones(900),np.zeros(900) ))
Y_test = np.concatenate(( np.ones(100), np.zeros(100) ))

## X_train, X_test, Y_train, Y_test are your datasets to work with ####
```

Instructions to follow for learning a Baeysian classifier: *(Code the formulae for estimating the different parameters yourself)*

a)Utilize the training dataset to estimate the means($\hat{\mu_+}$,$\hat{\mu_-}$) and variances($\hat{\sigma_+^2}$, $\hat{\sigma_-^2}$) for both positive and negative classes

b)Estimate the prior probability: $P(Y=1) \longrightarrow$ which could be referred to as: $\hat{a}$

c)Estimate the classifier funtion/posterior probability: $P(Y=1|X=x) \longrightarrow$ which could be referred to as $\hat{\eta(x)}$

d)Find out the threshold value(x^*) for classification by equating the estimated classifier function($\hat{\eta(x)}$) with threshold probability of 0.5

e)Classify the test dataset into the two classes using this threshold value(x^*) and find out the **accuracy** of the prediction

Return back: $\hat{\mu_+}$, $\hat{\mu_-}$, $\hat{\sigma_+^2}$, $\hat{\sigma_-^2}$, $\hat{a}$, x^* and accuracy from the code written

Hint: $X|Y_{Y=0} \sim \mathcal{N}(\mu_-, \sigma_-^2)$ *implies* $P_{X|Y=0} = \mathcal{N}(\mu_-, \sigma_-^2)$

In [10]:

```
## write your code here.
#Finding mean and variance

def mean(x):
    mean = np.sum(x)/len(x)
    return mean

def variance(x):
    mu = mean(x)
    var=0
    for i in x:
        var = var + (((i - mu)**2)/(len(x)-1))
    return var[0]



mean_pos = mean(X_train[Y_train==1])
var_pos = variance(X_train[Y_train==1])


mean_neg = mean(X_train[Y_train==0])
var_neg = variance(X_train[Y_train==0])

print("Mean for positive class = %.3f \nVariance for positive class = %.3f \nMean for n
egative class = %.3f \nVariance for negative class = %.3f" %(mean_pos,var_pos,mean_neg,
var_neg))
```

```
Mean for positive class = 2.027
Variance for positive class = 0.974
Mean for negative class = 3.967
Variance for negative class = 1.000
```

In [11]:

```
import math

a_hat = len(X_train[Y_train==1])/len(X_train)
a_hat
```

Out[11]:

```
0.5
```

In [12]:

```
#Posterior probability - eta_hat
def eta_hat(x, mean_pos, var_pos, mean_neg, var_neg, a_hat):
    pdf_pos = (1/(var_pos*(math.sqrt(2*math.pi))))*(math.exp((-1/2)*(((x-mean_pos)/var_
pos)**2)))
    pdf_neg = (1/(var_neg*(math.sqrt(2*math.pi))))*(math.exp((-1/2)*(((x-mean_neg)/var_
neg)**2)))

    return a_hat*pdf_pos/(a_hat*pdf_pos + ((1-a_hat)*pdf_neg))
```

In [13]:

```
probs={}
for i in X_train:
    probs[i[0]] = eta_hat(i, mean_pos, var_pos, mean_neg, var_neg, a_hat)

sort_probs = sorted(probs.items(), key=lambda x: x[1], reverse=False)
for k in range(len(probs)):
    if sort_probs[k][1]>=0.5:
        threshold = sort_probs[k][0]
        break

print('Threshold value of x: x* = %.4f' %threshold)
```

```
Threshold value of x: x* = 2.9963
```

In [14]:

```
y_pred_test=[]
for x in X_test:
    if x>=threshold:
        y_pred_test.append(0)
    else:
        y_pred_test.append(1)

from sklearn.metrics import confusion_matrix

CM = confusion_matrix(Y_test,y_pred_test)
print('Confusion matrix:\n',CM)
test_acc = (CM[1][1]+CM[0][0])/(CM[0][0]+CM[0][1]+CM[1][0]+CM[1][1])
print("Test Accuracy: %.2f" %test_acc)
```

```
Confusion matrix:
 [[85 15]
 [13 87]]
Test Accuracy: 0.86
```

Write results here

In [15]:

```
print("Mean for positive class = %.3f \nVariance for positive class = %.3f \nMean for n
egative class = %.3f \nVariance for negative class = %.3f" %(mean_pos,var_pos,mean_neg,
var_neg))
print('Prior probability a_hat =',a_hat)
print('Threshold value = %.3f' %threshold)
print("Test Accuracy: %.2f" %test_acc)
```

```
Mean for positive class = 2.027
Variance for positive class = 0.974
Mean for negative class = 3.967
Variance for negative class = 1.000
Prior probability a_hat = 0.5
Threshold value = 2.996
Test Accuracy: 0.86
```

From Discrete distribution of data

Unlike the first exercise for learning the Naive Bayes' classifier where we dealt with continuous distribution of data, here you need to work with discrete data, which means finding Probability Mass Distribution(PMF).

Age	Income	Status	Buy
<=20	low	students	yes
<=20	high	students	yes
<=20	medium	students	no
<=20	medium	married	no
<=20	high	married	yes
21-30	low	married	yes
21-30	low	married	no
21-30	medium	students	no
21-30	high	students	yes
>30	high	married	no
>30	high	married	yes
>30	medium	married	yes
>30	medium	married	no
>30	medium	students	no

Consider the train dataset above. Take any random datapoint (X_i) where
$X_i = (X_{i,1} = Age, X_{i,2} = Income, X_{i,3} = Status)$ and its corresponding label

($Y_i = Buy$). A "yes" in Buy corresponds to label-1 and a "no" in Buy corresponds to label-0.

Instructions to follow for learning a Baeysian classifier: *(Code the formulae for estimating the different parameters yourself)*
a)Estimate the prior probability: $P(Y = 1) \longrightarrow$ which could be referred to as: $\hat{a}$
b)Estimate the likelihood for each feature: $P(X_{i,j} = x|Y = y_i)$, where i=datapoint counter, $j \in \{1, 2, 3\}$ and $y_i \in \{0, 1\}$
c)Estimate the total likelihood: $P(X_i = x|Y = y_i)$
d)Calculate the posterior probability: $P(Y = 1|X_i = x_{test})$ = p_{test} where
$x_{test} = (Age = 21 - 30, Income = medium, Status = married)$

Return back: $\hat{a}$, total likelihood and p_{test}

In [16]:

```
## write your code here.
discrete_data = pd.DataFrame([['<=20' ,  'low'    , 'students', 'yes'],
['<=20' ,  'high'   , 'students', 'yes'],
['<=20' , 'medium' , 'students', 'no'],
['<=20' , 'medium' , 'married' , 'no'],
['<=20' ,  'high'   , 'married' , 'yes'],
['21-30',  'low'    , 'married' , 'yes'],
['21-30',  'low'    , 'married' , 'no'],
['21-30', 'medium' , 'students', 'no'],
['21-30',  'high'   , 'students', 'yes'],
['>30' ,  'high'   , 'married' , 'no'],
['>30' ,  'high'   , 'married' , 'yes'],
['>30' , 'medium' , 'married' , 'yes'],
['>30' , 'medium' , 'married' , 'no'],
['>30' , 'medium' , 'students', 'no']], columns=['Age'  , 'Income' , 'Status'  , 'Buy'
])
```

Write results here

In [17]:

```
#Buy = yes ==> Y=1
a_hat = len(discrete_data[discrete_data['Buy']=='yes'])/len(discrete_data)
a_hat
```

Out[17]:

```
0.5
```

In [33]:

```
#Xi,1=Age, Xi,2=Income, Xi,3=Status

y1 = len(discrete_data[discrete_data['Buy']=='yes'])
y0 = len(discrete_data[discrete_data['Buy']=='no'])

#Age <=21 : 0; Age 21-30 : 1; Age>30: 2
age_y1 = [0,0,0]
age_y1[0] = len(discrete_data[(discrete_data['Age']=='<=20') & (discrete_data['Buy']==
'yes')])
age_y1[1] = len(discrete_data[(discrete_data['Age']=='21-30') & (discrete_data['Buy']==
'yes')])
age_y1[2] = len(discrete_data[(discrete_data['Age']=='>30') & (discrete_data['Buy']=='y
es')])

age_y0 = [0,0,0]
age_y0[0] = len(discrete_data[(discrete_data['Age']=='<=20') & (discrete_data['Buy']==
'no')])
age_y0[1] = len(discrete_data[(discrete_data['Age']=='21-30') & (discrete_data['Buy']==
'no')])
age_y0[2] = len(discrete_data[(discrete_data['Age']=='>30') & (discrete_data['Buy']=='n
o')])

print('P(Xi,1 = <=20 | Y=1) = %.4f' %(age_y1[0]/y1))
print('P(Xi,1 = 21-30 | Y=1) = %.4f' %(age_y1[1]/y1))
print('P(Xi,1 = >30 | Y=1) = %.4f' %(age_y1[2]/y1))
print('P(Xi,1 = <=20 | Y=0) = %.4f' %(age_y0[0]/y0))
print('P(Xi,1 = 21-30 | Y=0) = %.4f' %(age_y0[1]/y0))
print('P(Xi,1 = >30 | Y=0) = %.4f' %(age_y0[2]/y0))


#Income low:0, medium:1, high:2
inc_y1 = [0,0,0]
inc_y1[0] = len(discrete_data[(discrete_data['Income']=='low') & (discrete_data['Buy']=
='yes')])
inc_y1[1] = len(discrete_data[(discrete_data['Income']=='medium') & (discrete_data['Bu
y']=='yes')])
inc_y1[2] = len(discrete_data[(discrete_data['Income']=='high') & (discrete_data['Buy']
=='yes')])

inc_y0 = [0,0,0]
inc_y0[0] = len(discrete_data[(discrete_data['Income']=='low') & (discrete_data['Buy']=
='no')])
inc_y0[1] = len(discrete_data[(discrete_data['Income']=='medium') & (discrete_data['Bu
y']=='no')])
inc_y0[2] = len(discrete_data[(discrete_data['Income']=='high') & (discrete_data['Buy']
=='no')])

print('\nP(Xi,2 = low | Y=1) = %.4f' %(inc_y1[0]/y1))
print('P(Xi,2 = medium | Y=1) = %.4f' %(inc_y1[1]/y1))
print('P(Xi,2 = high | Y=1) = %.4f' %(inc_y1[2]/y1))
print('P(Xi,2 = low | Y=0) = %.4f' %(inc_y0[0]/y0))
print('P(Xi,2 = medium | Y=0) = %.4f' %(inc_y0[1]/y0))
print('P(Xi,2 = high | Y=0) = %.4f' %(inc_y0[2]/y0))

#Status students:0 ; married:1
status_y1 = [0,0]
status_y1[0] = len(discrete_data[(discrete_data['Status']=='students') & (discrete_data
['Buy']=='yes')])
status_y1[1] = len(discrete_data[(discrete_data['Status']=='married') & (discrete_data[
```

```
'Buy']=='yes')])

status_y0 = [0,0]
status_y0[0] = len(discrete_data[(discrete_data['Status']=='students') & (discrete_data
['Buy']=='no')])
status_y0[1] = len(discrete_data[(discrete_data['Status']=='married') & (discrete_data[
'Buy']=='no')])

print('\nP(Xi,3 = student | Y=1) = %.4f' %(status_y1[0]/y1))
print('P(Xi,3 = married | Y=1) = %.4f' %(status_y1[1]/y1))
print('P(Xi,3 = student | Y=0) = %.4f' %(status_y0[0]/y0))
print('P(Xi,3 = married | Y=0) = %.4f' %(status_y0[1]/y0))
```

```
P(Xi,1 = <=20 | Y=1) = 0.4286
P(Xi,1 = 21-30 | Y=1) = 0.2857
P(Xi,1 = >30 | Y=1) = 0.2857
P(Xi,1 = <=20 | Y=0) = 0.2857
P(Xi,1 = 21-30 | Y=0) = 0.2857
P(Xi,1 = >30 | Y=0) = 0.4286

P(Xi,2 = low | Y=1) = 0.2857
P(Xi,2 = medium | Y=1) = 0.1429
P(Xi,2 = high | Y=1) = 0.5714
P(Xi,2 = low | Y=0) = 0.1429
P(Xi,2 = medium | Y=0) = 0.7143
P(Xi,2 = high | Y=0) = 0.1429

P(Xi,3 = student | Y=1) = 0.4286
P(Xi,3 = married | Y=1) = 0.5714
P(Xi,3 = student | Y=0) = 0.4286
P(Xi,3 = married | Y=0) = 0.5714
```

In [38]:

```
#Total likelihood
prob1sum=0
prob2sum=0
for i in range(len(age_y0)):
    for j in range(len(inc_y0)):
        for k in range(len(status_y0)):
            prob_1 = (age_y1[i]*inc_y1[j]*status_y1[k])/(y1**3)
            prob_0 = (age_y0[i]*inc_y0[j]*status_y0[k])/(y1**3)
            print('P(age =',i,', income =', j,', status =', k,'| Y=1): %.4f' %prob_1)
            print('P(age =',i,', income =', j,', status =', k,'| Y=0): %.4f' %prob_0)


#Age <=21 : 0; Age 21-30 : 1; Age>30: 2
#Income low:0, medium:1, high:2
#Status students:0 ; married:1
```

```
P(age = 0 , income = 0 , status = 0 | Y=1): 0.0525
P(age = 0 , income = 0 , status = 0 | Y=0): 0.0175
P(age = 0 , income = 0 , status = 1 | Y=1): 0.0700
P(age = 0 , income = 0 , status = 1 | Y=0): 0.0233
P(age = 0 , income = 1 , status = 0 | Y=1): 0.0262
P(age = 0 , income = 1 , status = 0 | Y=0): 0.0875
P(age = 0 , income = 1 , status = 1 | Y=1): 0.0350
P(age = 0 , income = 1 , status = 1 | Y=0): 0.1166
P(age = 0 , income = 2 , status = 0 | Y=1): 0.1050
P(age = 0 , income = 2 , status = 0 | Y=0): 0.0175
P(age = 0 , income = 2 , status = 1 | Y=1): 0.1399
P(age = 0 , income = 2 , status = 1 | Y=0): 0.0233
P(age = 1 , income = 0 , status = 0 | Y=1): 0.0350
P(age = 1 , income = 0 , status = 0 | Y=0): 0.0175
P(age = 1 , income = 0 , status = 1 | Y=1): 0.0466
P(age = 1 , income = 0 , status = 1 | Y=0): 0.0233
P(age = 1 , income = 1 , status = 0 | Y=1): 0.0175
P(age = 1 , income = 1 , status = 0 | Y=0): 0.0875
P(age = 1 , income = 1 , status = 1 | Y=1): 0.0233
P(age = 1 , income = 1 , status = 1 | Y=0): 0.1166
P(age = 1 , income = 2 , status = 0 | Y=1): 0.0700
P(age = 1 , income = 2 , status = 0 | Y=0): 0.0175
P(age = 1 , income = 2 , status = 1 | Y=1): 0.0933
P(age = 1 , income = 2 , status = 1 | Y=0): 0.0233
P(age = 2 , income = 0 , status = 0 | Y=1): 0.0350
P(age = 2 , income = 0 , status = 0 | Y=0): 0.0262
P(age = 2 , income = 0 , status = 1 | Y=1): 0.0466
P(age = 2 , income = 0 , status = 1 | Y=0): 0.0350
P(age = 2 , income = 1 , status = 0 | Y=1): 0.0175
P(age = 2 , income = 1 , status = 0 | Y=0): 0.1312
P(age = 2 , income = 1 , status = 1 | Y=1): 0.0233
P(age = 2 , income = 1 , status = 1 | Y=0): 0.1749
P(age = 2 , income = 2 , status = 0 | Y=1): 0.0700
P(age = 2 , income = 2 , status = 0 | Y=0): 0.0262
P(age = 2 , income = 2 , status = 1 | Y=1): 0.0933
P(age = 2 , income = 2 , status = 1 | Y=0): 0.0350
1.0000000000000002 0.9999999999999999
```

Age <=21 : 0; Age 21-30 : 1; Age>30: 2

Income low:0, medium:1, high:2

In [45]:

```
#Posterior Probability
#xtest=(Age=21-30, Income=medium, Status=married)
xtest = [1,1,1]
#P(yes|xtest) = P(age=21-30|yes) x P(age=21-30|yes) x P(age=21-30|yes) x P(yes)
numerator = ((age_y1[xtest[0]]/y1)*(inc_y1[xtest[1]]/y1)*(status_y1[xtest[2]]/y1)*(a_ha
t))
denominator = (len(discrete_data[discrete_data['Age']=='21-30'])*len(discrete_data[disc
rete_data['Income']=='medium'])*len(discrete_data[discrete_data['Status']=='married']))
/(len(discrete_data)**3)

ptest = numerator/denominator


print('P(Y=1 | X=X_test): %.4f' %ptest)
print('P(Y=0 | X=X_test): %.4f' %ptest_y0)
print('P(Y=1 | X=X_test): %.4f < P(Y=0 | X=X_test): %.4f => Our prediction is Y=0 or "n
o"' %(ptest, 1-ptest))
```

```
P(Y=1 | X=X_test): 0.1667
P(Y=0 | X=X_test): 0.8333
P(Y=1 | X=X_test): 0.1667 < P(Y=0 | X=X_test): 0.8333 => Our prediction is
Y=0 or "no"
```

p_test = 0.0117

a_hat = 0.5

In [ ]:

```

```