# Sheet 7. Basharat Mubashir Ahmed

Thursday, 23. December 2021 12:29

## Task 1

(a)

We construct the table:-

| $n$ | $x_n$ | $y_n$ | $H_D\{a,b,d\}$ | $H_D\{f\}$ |
|---|---|---|---|---|
| 1 | $\{a,b,d,e\}$ | 3 | 1 | 5 |
| 2 | $\{c,d\}$ | 1 | 3 | 3 |
| 3 | $\{a\}$ | 2 | 2 | 2 |
| 4 | $\{a,f,g\}$ | 3 | 4 | 2 |
| 5 | $\{e,f\}$ | 4 | 5 | 1 |

Given, $f_D(\{a,b,d\}) = 2$
$f_D(\{f\}) = 3$

The observation indicates that $k=3$ was used as seen.

for $\{a,b,d\}$, 3 NN based on Hamming distance are:-

(i) $\{a,b,d,e\}$, $y_1 = 3$ (ii) $\{a\}$, $y_3 = 2$ (iii) $\{c,d\}$, $y_2 = 1$

Hence, prediction of $\{a,b,d\}$ with $k=3$ will be:-

$\bar{y} = \frac{1}{3}(3+2+1) = 2$ which is the given value of $f_D(\{a,b,d\}) = 2$.

For $\{f\}$, 3-NN based on Hamming distance are:-

(i) $\{e,f\}$, $y_5 = 4$ (ii) $\{a,f,g\}$, $y_4 = 3$ (iii) $\{a\}$, $y_3 = 2$

Hence, prediction of $\{f\}$ with $k=3$ will be:-

$\bar{y} = \frac{1}{3}(4+2+3) = 3$ which is the given value of $f_D(\{f\}) = 3$

Thus, with $k=3$, these predictions are indeed obtained.

b) We construct the table:-

| $n$ | $x_n$ | $y_n$ | $J_D\{a,b,d\}$ | $J_D\{f\}$ |
|---|---|---|---|---|
| 1 | $\{a,b,d,e\}$ | 3 | $\frac{3}{4} = 0.75$ | $\frac{0}{5} = 0$ |
| 2 | $\{c,d\}$ | 1 | $\frac{1}{4} = 0.25$ | $\frac{0}{3} = 0$ |
| 3 | $\{a,\}$ | 2 | $\frac{1}{3} = 0.33$ | $\frac{0}{2} = 0$ |
| 4 | $\{a,f,g\}$ | 3 | $\frac{1}{5} = 0.2$ | $\frac{1}{3} = 0.33$ |
| 5 | $\{e,f\}$ | 4 | $\frac{0}{5} = 0$ | $\frac{1}{2} = 0.25$ |

Given, $f_D(\{a,b,d\}) = 2.5$ and $f_D(\{f\}) = 3.5$.
The observation indicates that $k=2$ was used.

For $k=2$, Jaccard similarity with $\{a,b,d\}$ is:-

(i) $\{a,b,d,e\}$ with $y_1 = 3$ (ii) $\{a\}$ with $y_3 = 2$

Hence, prediction for $\{a,b,d\}$ will be:-

$\bar{y} = \frac{1}{2}(3+2) = 2.5$ which is the given value of $f_D(\{a,b,d\})$

For $k=2$, Jaccard similarity with $\{f\}$ is:-

(i) $\{a,f,g\}$ with $y_4 = 3$ (ii) $\{e,f\}$ with $y_5 = 4$

Hence, prediction for $\{f\}$ will be:-

$\bar{y} = \frac{1}{2}(3+4) = 3.5$ which is the given value of $f_D(\{f\})$.

Thus, with $k=2$, these predictions are indeed obtained.

## Task 3

We use the recurrence:-

$$D(i,j) = \begin{cases} D(i-1,j-1), & x_i = y_i \\ 1+\min[D(i-1,j)+D(i,j-1), D(i-1,j-1)], & x_i \neq y_i \end{cases}$$

with base case of

$$D(0,j) = j, \quad D(i,0) = i$$

| | | e | x | e | c | u | t | i | o | n |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| i | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 6 | 7 | 8 |
| n | 2 | 2 | 2 | 3 | 4 | 5 | 6 | 7 | 7 | 7 |
| t | 3 | 3 | 3 | 3 | 4 | 5 | $\boxed{5}$ | 6 | 7 | 8 |
| e | 4 | 3 | 4 | 3 | 4 | 5 | 6 | 6 | 7 | 8 |
| n | 5 | 4 | 4 | 4 | 4 | 5 | 6 | 7 | 7 | 7 |
| t | 6 | 5 | 5 | $\boxed{5}$ | 5 | 5 | 5 | 6 | 7 | 8 |
| i | 7 | 6 | 6 | 6 | 6 | 6 | 6 | 5 | 6 | 7 |
| o | 8 | 7 | 7 | 7 | 7 | 7 | 7 | 6 | 5 | 6 |
| n | 9 | 8 | 8 | 8 | 8 | 8 | 8 | 7 | 6 | $\boxed{5}$ |

⇒ Levenshtein distance between intention and execution is 5.
⇒ Levenshtein distance between intent and exe is 5.
⇒ Levenshtein distance between int and execut is 5.

# Basharat Mubashir Ahmed.

# 312093

# Machine Learning

# Question 2 - Visualisation of decision surface of KNN

In [1]:

```
# We import necessary libraries
# Please do not use scikit-learn or any other package. Implement K-NN classification yourse
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
from matplotlib.colors import ListedColormap
```

In [2]:

```
# Here we read the provided ushape.csv file
# We have retained only a small number of rows to ensure computational easiness and clear v
df = pd.read_csv('ushape.csv',names=['X','Y', 'C'], header=0)
ndf = df.drop(df.index[:50])
ndf = df.drop(df.index[:-10])
df
```

Out[2]:

| | X | Y | C |
|---|---|---|---|
| **0** | 0.0316 | 0.9870 | 0.0 |
| **1** | 2.1200 | -0.0462 | 1.0 |
| **2** | 0.8820 | -0.0758 | 0.0 |
| **3** | -0.0551 | -0.0373 | 1.0 |
| **4** | 0.8300 | -0.5390 | 1.0 |
| **...** | ... | ... | ... |
| **95** | 1.7000 | 0.5880 | 1.0 |
| **96** | 0.2190 | -0.6530 | 1.0 |
| **97** | 0.9530 | -0.4200 | 1.0 |
| **98** | -1.3200 | 0.4230 | 0.0 |
| **99** | -1.3000 | 0.1840 | 0.0 |

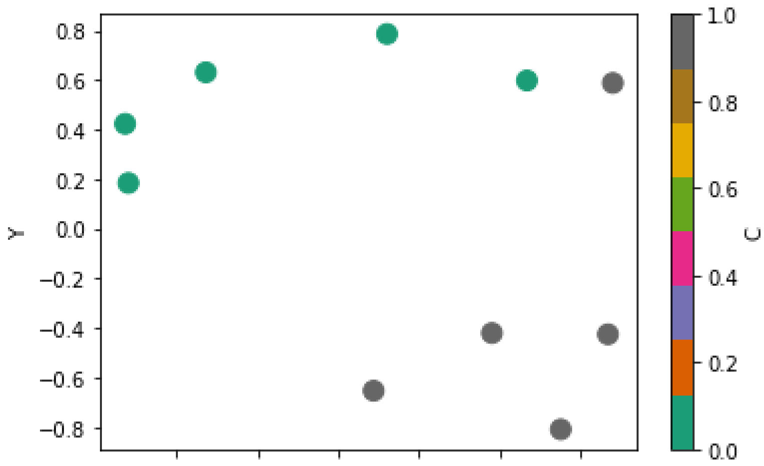
100 rows × 3 columns

In [3]:

```
# Let us see how the data looks like
ndf.plot.scatter('X','Y',c='C', s=100, colormap='Dark2')
```

Out[3]:

```
<AxesSubplot:xlabel='X', ylabel='Y'>
```




### Computing the L2 distance between two set of points.

In [4]:

```
def l2(x1,y1, x2, y2):
    distance = np.sqrt((x1-x2)**2 + (y1-y2)**2)
    return distance
```

### Computing distance between a test point and the training points. We also append the class label to the distance array making it a pair of distance and the class label.

In [5]:

```
def distance(x_test, y_test):
    distances = list()
    for index, row in ndf.iterrows():
        d = l2(row['X'], row['Y'], x_test, y_test)
        distances.append((d,row['C']))
    return distances
```

### Assigns the class label to a test poing using majority vote by K-NN classification.

In [6]:

```
def knn_classification(x_test, y_test, k):
    dist=distance(x_test,y_test)
    dist=sorted(dist)[:k]
    labels=list(zip(*dist))[1]
    pred=Counter(labels).most_common(k)[0][0]
    return pred
```

### Testing the implementation against few test points.

In [7]:

```
compare=[]
for i in range(15):
    pair=df.loc[i]
    px,py,c=pair[0],pair[1],pair[2]
    label=knn_classification(px,py,2)
    compare.append({'Actual':c,'Predicted':label})
compare
```

Out[7]:

```
[{'Actual': 0.0, 'Predicted': 0.0},
 {'Actual': 1.0, 'Predicted': 1.0},
 {'Actual': 0.0, 'Predicted': 1.0},
 {'Actual': 1.0, 'Predicted': 1.0},
 {'Actual': 1.0, 'Predicted': 1.0},
 {'Actual': 1.0, 'Predicted': 1.0},
 {'Actual': 0.0, 'Predicted': 0.0},
 {'Actual': 1.0, 'Predicted': 1.0},
 {'Actual': 1.0, 'Predicted': 0.0},
 {'Actual': 1.0, 'Predicted': 1.0},
 {'Actual': 1.0, 'Predicted': 0.0},
 {'Actual': 1.0, 'Predicted': 1.0},
 {'Actual': 1.0, 'Predicted': 1.0},
 {'Actual': 0.0, 'Predicted': 0.0},
 {'Actual': 0.0, 'Predicted': 0.0}]
```

### Plotting the decision surface for the two classes for given K value.

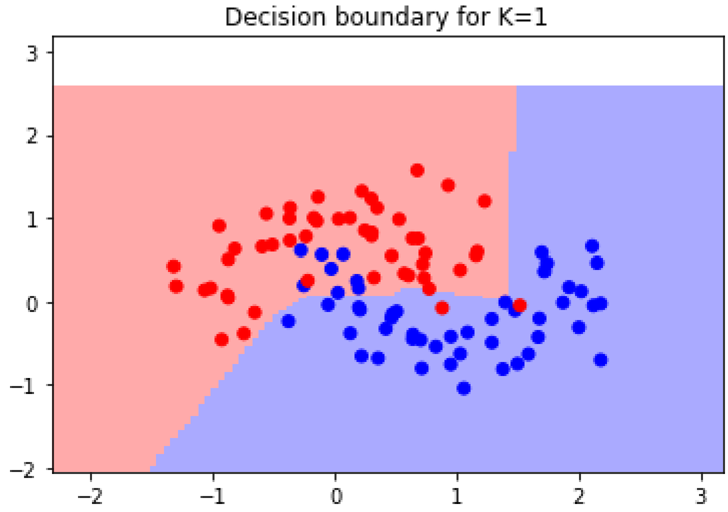
In [8]:

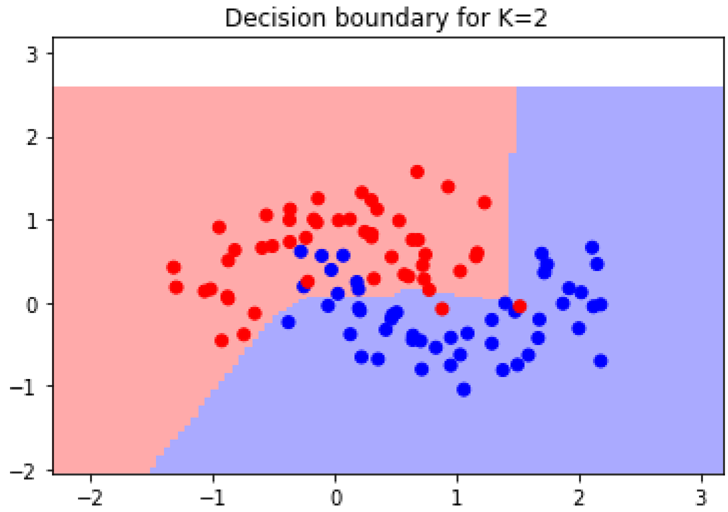
```
def plot_decision_surface(k):
    light=ListedColormap(['#FFAAAA', '#AAAAFF'])
    bold=ListedColormap(['#FF0000','#0000FF'])
    x=df[['X','Y']].to_numpy()
    xmin,xmax=x[:, 0].min()-1,x[:, 0].max()+1
    ymin,ymax=x[:, 1].min()-1,x[:, 1].max()+1
    XX,YY=np.meshgrid(np.linspace(xmin,xmax,100),np.linspace(ymin,ymax,100))
    ZZ=[]
    for xx,yy in zip(XX.ravel(),YY.ravel()):
        ZZ.append(knn_classification(xx,yy,k))
    ZZ=np.array(ZZ).reshape(XX.shape)
    plt.pcolormesh(XX, YY, ZZ,cmap=light,shading='auto')
    plt.scatter(x[:,0], x[:,1],c=df['C'].to_numpy(),cmap=bold)
    plt.xlim(XX.min(), XX.max());plt.ylim(YY.min(), XX.max())
    plt.title('Decision boundary for K=%d'%k)
```

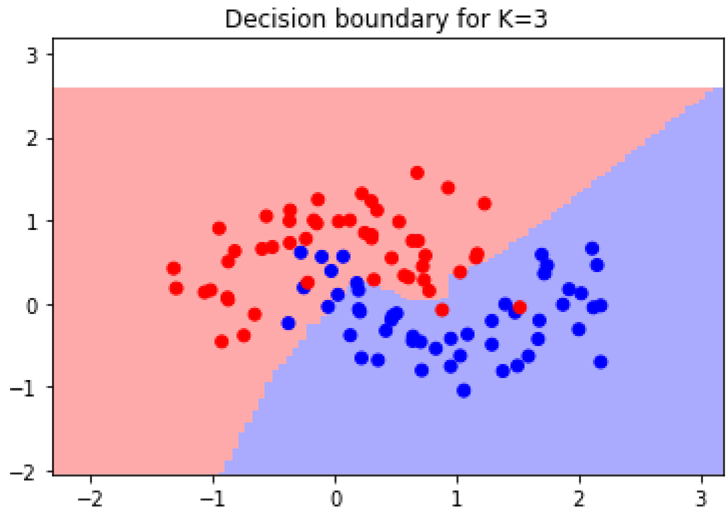
### Plotting the decision boundary for K=1,2 and 3

In [9]:

```
K=[1,2,3]
for k in K:
    plot_decision_surface(k)
    plt.figure()
```




```
<Figure size 432x288 with 0 Axes>
```

## We note that as the value of K increases, the decision boundary gets smoother.

# Question 3. Computing the Levenshtein distance.

### Building the dp matrix using bottom up Dynamic Programming approach

In [10]:

```
def Levenshtein(s1,s2):
    l1,l2=len(s1),len(s2)
    dp=[[-1 for x in range(l2+1)] for x in range(l1+1)]
    for i in range(l1+1):
        for j in range(l2+1):
            if i==0:
                dp[i][j]=j
            elif j==0:
                dp[i][j]=i
            elif s1[i-1]==s2[j-1]:
                dp[i][j]=dp[i-1][j-1]
            else:
                dp[i][j]=1+min(dp[i-1][j],dp[i][j-1],dp[i-1][j-1])
    return dp
```

### Returning the full DP matrix computed.

In [11]:

```
s1="intention"
s2="execution"
matrix=Levenshtein(s1,s2)
print('The DP matrix is: ')
matrix
```

```
The DP matrix is: 
```

Out[11]:

```
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 [1, 1, 2, 3, 4, 5, 6, 6, 7, 8],
 [2, 2, 2, 3, 4, 5, 6, 7, 7, 7],
 [3, 3, 3, 3, 4, 5, 5, 6, 7, 8],
 [4, 3, 4, 3, 4, 5, 6, 6, 7, 8],
 [5, 4, 4, 4, 4, 5, 6, 7, 7, 7],
 [6, 5, 5, 5, 5, 5, 5, 6, 7, 8],
 [7, 6, 6, 6, 6, 6, 6, 5, 6, 7],
 [8, 7, 7, 7, 7, 7, 7, 6, 5, 6],
 [9, 8, 8, 8, 8, 8, 8, 7, 6, 5]]
```

### Last entry of the matrix gives the Levenshtein distance between "intention" and execution which is 5.

### Levenshtein distance between "intent" and "exe" is the entry (3,6) which is 5.

### Levenshtein distance between "int" and "execut" is the entry (6,3) which is 5.