K-NN Classification Implementation

In the following diagram let blue circles indicate positive examples and orange squares indicate negative examples. We want to use k-NN algorithm for classifying the points. If k=3, find the class of the point (6,6). Extend the same example for Distance-Weighted k-NN and Locally weighted Averaging.

In [5]:

```
import math
import pandas as pd
```

In [6]:

```
xdata=[2,4,4,4,6,6]
ydata=[4,2,4,6,2,4]
res  =[0,0,1,0,1,0]
```

In [26]:

```
def knn_classification(xtest, ytest):
    for i in range(len(xtest)):
        for j in range(len(xtrain)):
            xd=(xtest[i]-xtrain[j])**2
            yd=(ytest[i]-ytrain[j])**2
            d=math.sqrt(xd+yd)
            row=(xtest[i],ytest[i],xtrain[j],ytrain[j],trainclass[j],d)
            distances.append(row)

        l=[x[5] for x in distances if distances[0]==xtest[0] and distances[1]==ytest[0]]
        l.sort()
        topk=l[:3]
        topclasses=list()

        for i in topk:
            for j in distances:
                if i==j[5]:
                    topclasses.append(j[4])

        topclasses=topclasses[:k]
        pos=topclasses.count(1)
        neg=topclasses.count(0)

        cl=-1
        if pos>=neg:
            cl=1
        else:
            cl=0

        predclass.append(cl)
        print("Prediction for ("+str(xtest[i])+","+str(ytest[i])+"):",cl)
```

In [27]:

```
k=3
predclass=list()
distances=list()

xtrain=xdata[:4]
ytrain=ydata[:4]
xtest=xdata[4:]
ytest=ydata[4:]

trainclass=res[:4]
testclass=res[4:]

knn_classification(xtest, ytest)
```

```
Prediction for (6,2): 1
Prediction for (6,4): 1
```

In [28]:

```
hit=0
for i in range(len(testclass)):
    if testclass[i]==predclass[i]:
        hit=hit+1

n=len(testclass)
acc=hit/n
print("Accuracy Score:",acc)
```

```
Accuracy Score: 0.5
```

K-NN Classification for (6,6)

In [16]:

```
xtest=[6]
ytest=[6]
k=3
distances=list()

for i in range(len(xtest)):
    for j in range(len(xtrain)):
        xd=(xtest[i]-xtrain[j])**2
        yd=(ytest[i]-ytrain[j])**2
        d=math.sqrt(xd+yd)
        row=(xtest[i],ytest[i],xtrain[j],ytrain[j],trainclass[j],d)
    distances.append(row)

    l=[x[5] for x in distances if distances[0]==xtest[0] and distances[1]==ytest[0]]
    l.sort()
    topk=l[:3]
    topclasses=list()

    for i in topk:
        for j in distances:
            if i==j[5]:
                topclasses.append(j[4])

    topclasses=topclasses[:k]
    pos=topclasses.count(1)
    neg=topclasses.count(0)

    cl=-1
    if pos>neg:
        cl=1
    elif pos<neg:
        cl=0
    else:
        cl="Can be 0 or 1"

    print("Prediction for ("+str(xtest[i])+","+str(ytest[i])+"):",cl)
```

```
Prediction for (6,6): Can be 0 or 1
```

Weighted K-NN Classification

In [17]:

```
import math

xdata=[2,4,4,4,6,6]
ydata=[4,2,4,6,2,4]
res  =[0,0,1,0,1,0]

data=[[2,4,0],[4,2,0],[4,4,1],[4,6,0],[6,2,1],[6,4,0]]
```

```
df=pd.DataFrame(data,columns=['X-Coordinate','Y-Coordinate','Class'])

xtrain=xdata[:4]
ytrain=ydata[:4]
xtest=xdata[4:]
ytest=ydata[4:]
trainclass=res[:4]
testclass=res[4:]
```

In [18]:

```
k=3
predclass=list()
distances=list()
for i in range(len(xtest)):
    for j in range(len(xtrain)):
        xd=(xtest[i]-xtrain[j])**2
        yd=(ytest[i]-ytrain[j])**2
        d=math.sqrt(xd+yd)
        row=(d,trainclass[j])
        distances.append(row)
    distances = sorted(distances)[:k]

    freq1=0
    freq2=0

    for d in distances:
        if d[1]==0:
            freq1=freq1+(1/d[0])
        else:
            freq2=freq2+(1/d[0])

    if freq1>freq2:
        cl=0
    else:
        cl=1

    predclass.append(cl)
    print("Prediction for ("+str(xtest[i])+","+str(ytest[i])+"):",cl)

hit=0
for i in range(len(testclass)):
    if testclass[i]==predclass[i]:
        hit=hit+1

n=len(testclass)
acc=hit/n
print("Accuracy Score:",acc)
```

```
Prediction for (6,2): 0
Prediction for (6,4): 0
Accuracy Score: 0.5
```

Weighted K-NN Classification for (6,6)

In [30]:

```
xtest=[6]
ytest=[6]
k=3
distances=list()
for i in range(len(xtest)):
    for j in range(len(xtrain)):
        xd=(xtest[i]-xtrain[j])**2
        yd=(ytest[i]-ytrain[j])**2
        d=math.sqrt(xd+yd)
        row=(d,trainclass[j])
        distances.append(row)
    distances = sorted(distances)[:k]

    freq1=0
    freq2=0

    for d in distances:
        if d[1]==0:
            freq1=freq1+(1/d[0])
        else:
            freq2=freq2+(1/d[0])

    if freq1>freq2:
        cl=0
    else:
        cl=1

    print(freq1,freq2)
    print("Prediction for ("+str(xtest[i])+","+str(ytest[i])+"):",cl)
```

```
0.7236067977499789 0.35355339059327373
Prediction for (6,6): 0
```

K-NN Classification using Scikit-learn

In [31]:

```
import pandas as pd

data=[[2,4,0],[4,2,0],[4,4,1],[4,6,0],[6,2,1],[6,4,0]]
df=pd.DataFrame(data,columns=['X-Coordinate','Y-Coordinate','Class'])
df

#1 indicates Positive Samples and 0 indicates negative samples
```

Out[31]:

	X-Coordinate	Y-Coordinate	Class
0	2	4	0
1	4	2	0
2	4	4	1
3	4	6	0
4	6	2	1
5	6	4	0

In [33]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

In [34]:

```
X=df.drop('Class',axis=1)
y=df.Class

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)

ypred1 = knn.predict(X_test)
accuracy1=accuracy_score(ypred1,y_test)
print("Accuracy:",accuracy_score(ypred1,y_test))

test=np.array([6,6])
pred=knn.predict(test.reshape(1,-1))
print("Prediction for sample (6,6):",pred)
```

```
Accuracy: 0.5
```

```
Prediction for sample (6,6): [0]
```

Weighted K-NN Classification for K-NN

In [35]:

```
distanceknn = KNeighborsClassifier(n_neighbors=3,weights='distance')
distanceknn.fit(X,y)

ypred2 = distanceknn.predict(X_test)
accuracy2=accuracy_score(ypred2,y_test)
print("Accuracy:",accuracy_score(ypred2,y_test))

test=np.array([6,6])
ypred=distanceknn.predict(test.reshape(1,-1))
print("Prediction for sample (6,6):",ypred)
```

```
Accuracy: 1.0
Prediction for sample (6,6): [0]
```

Weighted Average Accuracy

In [36]:

```
averagepred=((0.5*accuracy1+0.5*accuracy2))
print("Average Accuracy:",averagepred)

averagepred=((0.4*accuracy1+0.6*accuracy2))
print("Weighted average Accuracy:",averagepred)
```

```
Average Accuracy: 0.75
Weighted average Accuracy: 0.8
```

In [ ]:

```

```