K Nearest Neighbors

A supervised machine learning method that assess the euclidean distance between two data point. The unknown point is then assigned the category of the nearest points in the data set

In [326]:

```
import pandas as pd
import math
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score, precision_score, recall_score
import seaborn as sns
from sklearn import metrics



BMI = {
        'Weight':['51','62','69','64','65','56','58','57','55'],
        'Height':['167','182','176','173','172','174','169','173','170'],
        'Class':['Underweight','Normal','Normal','Normal','Normal','Underweigh
t','Normal','Normal','Normal']
}

BMIDF = pd.DataFrame(BMI)
BMIDF
```

Out[326]:

	Weight	Height	Class
0	51	167	Underweight
1	62	182	Normal
2	69	176	Normal
3	64	173	Normal
4	65	172	Normal
5	56	174	Underweight
6	58	169	Normal
7	57	173	Normal
8	55	170	Normal

Say we want to predict a new person with a weight of 57 and a height of 170

We calculate the eudcledian distance of this new unknow data with each point

sqrt(x - a)2 + (y - b)2

i.e.

point 1 sqrt(57-51)2 + (170-167)2 = 6.7 point 2 sqrt(57-62)2 + (170-182)2 = 13.

Continue for each point

Calculation to manually the eucledian distance

In [83]:
```
weight = BMIDF['Weight']
height = BMIDF['Height']



weightadjusted = [(57 - int(i))**2 for i in weight]
weightadjusted

heightadjusted = [(170 - int(i))**2 for i in height]
heightadjusted


eucledian distance = [math.sqrt(a + b) for a, b in zip(weightadjusted,heightadj
usted)]
eucledian distance
```

Out[83]:
```
[6.708203932499369,
 13.0,
 13.416407864998739,
 7.615773105863909,
 8.246211251235321,
 4.123105625617661,
 1.4142135623730951,
 3.0,
 2.0]
```

In [ ]:
```

```

Add the Eucledian Distance column to the original data set

In [84]:
```
BMI["euclediandistance"] = euclediandistance
BMIDFFinal = pd.DataFrame(BMI)
BMIDFFinal
```

Out[84]:

	Weight	Height	Class	euclediandistance
0	51	167	Underweight	6.708204
1	62	182	Normal	13.000000
2	69	176	Normal	13.416408
3	64	173	Normal	7.615773
4	65	172	Normal	8.246211
5	56	174	Underweight	4.123106
6	58	169	Normal	1.414214
7	57	173	Normal	3.000000
8	55	170	Normal	2.000000

According to the table with a K nearest neighbor to 3 ie the closest 3 neighbours in regards to distance. The closes points would be rows 6 to 8 with values being 1.4,2 and 3. Therefore assigning the class to the new data as Normal in weght

Diabetes Data Set

1. Opens the csv
2. Replaces all the columns that are supposed to have values with NAN
3. Finds the mean of each column skipping over the NAN values
4. then replaces the NAN values with the respective means

In [87]:

```
diabetes = pd.read_csv('C:/Users/mrjod/Desktop/Sample Data Sets for Data Scien
ce and STudying/Diabetes Data/diabetes.csv')


count = (diabetes['Age']==0).sum()

diabetes_no_zeros = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
'BMI']

for columns in diabetes_no_zeros:
    diabetes[columns] = diabetes[columns].replace(0,np.NAN)
    mean = diabetes[columns].mean(skipna = True)
    diabetes[columns] = diabetes[columns].replace(np.NAN, mean)

diabetes
```

Out[87]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
0	6	148.0	72.000000	35.00000	155.548223	33.600000	
1	1	85.0	66.000000	29.00000	155.548223	26.600000	
2	8	183.0	64.000000	29.15342	155.548223	23.300000	
3	1	89.0	66.000000	23.00000	94.000000	28.100000	
4	0	137.0	40.000000	35.00000	168.000000	43.100000	
5	5	116.0	74.000000	29.15342	155.548223	25.600000	
6	3	78.0	50.000000	32.00000	88.000000	31.000000	
7	10	115.0	72.405184	29.15342	155.548223	35.300000	
8	2	197.0	70.000000	45.00000	543.000000	30.500000	
9	8	125.0	96.000000	29.15342	155.548223	32.457464	
10	4	110.0	92.000000	29.15342	155.548223	37.600000	
11	10	168.0	74.000000	29.15342	155.548223	38.000000	
12	10	139.0	80.000000	29.15342	155.548223	27.100000	
13	1	189.0	60.000000	23.00000	846.000000	30.100000	
14	5	166.0	72.000000	19.00000	175.000000	25.800000	
15	7	100.0	72.405184	29.15342	155.548223	30.000000	
16	0	118.0	84.000000	47.00000	230.000000	45.800000	
17	7	107.0	74.000000	29.15342	155.548223	29.600000	
18	1	103.0	30.000000	38.00000	83.000000	43.300000	
19	1	115.0	70.000000	30.00000	96.000000	34.600000	
20	3	126.0	88.000000	41.00000	235.000000	39.300000	
21	8	99.0	84.000000	29.15342	155.548223	35.400000	
22	7	196.0	90.000000	29.15342	155.548223	39.800000	
23	9	119.0	80.000000	35.00000	155.548223	29.000000	
24	11	143.0	94.000000	33.00000	146.000000	36.600000	
25	10	125.0	70.000000	26.00000	115.000000	31.100000	
26	7	147.0	76.000000	29.15342	155.548223	39.400000	
27	1	97.0	66.000000	15.00000	140.000000	23.200000	
28	13	145.0	82.000000	19.00000	110.000000	22.200000	
29	5	117.0	92.000000	29.15342	155.548223	34.100000	
...	...	...	...	...	...	...	
738	2	99.0	60.000000	17.00000	160.000000	36.600000	
739	1	102.0	74.000000	29.15342	155.548223	39.500000	
740	11	120.0	80.000000	37.00000	150.000000	42.300000	
741	3	102.0	44.000000	20.00000	94.000000	30.800000	

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
742	1	109.0	58.000000	18.00000	116.000000	28.500000	
743	9	140.0	94.000000	29.15342	155.548223	32.700000	
744	13	153.0	88.000000	37.00000	140.000000	40.600000	
745	12	100.0	84.000000	33.00000	105.000000	30.000000	
746	1	147.0	94.000000	41.00000	155.548223	49.300000	
747	1	81.0	74.000000	41.00000	57.000000	46.300000	
748	3	187.0	70.000000	22.00000	200.000000	36.400000	
749	6	162.0	62.000000	29.15342	155.548223	24.300000	
750	4	136.0	70.000000	29.15342	155.548223	31.200000	
751	1	121.0	78.000000	39.00000	74.000000	39.000000	
752	3	108.0	62.000000	24.00000	155.548223	26.000000	
753	0	181.0	88.000000	44.00000	510.000000	43.300000	
754	8	154.0	78.000000	32.00000	155.548223	32.400000	
755	1	128.0	88.000000	39.00000	110.000000	36.500000	
756	7	137.0	90.000000	41.00000	155.548223	32.000000	
757	0	123.0	72.000000	29.15342	155.548223	36.300000	
758	1	106.0	76.000000	29.15342	155.548223	37.500000	
759	6	190.0	92.000000	29.15342	155.548223	35.500000	
760	2	88.0	58.000000	26.00000	16.000000	28.400000	
761	9	170.0	74.000000	31.00000	155.548223	44.000000	
762	9	89.0	62.000000	29.15342	155.548223	22.500000	
763	10	101.0	76.000000	48.00000	180.000000	32.900000	
764	2	122.0	70.000000	27.00000	155.548223	36.800000	
765	5	121.0	72.000000	23.00000	112.000000	26.200000	
766	1	126.0	60.000000	29.15342	155.548223	30.100000	
767	1	93.0	70.000000	31.00000	155.548223	30.400000	

768 rows × 9 columns

1. Uses iloc to get the get columns 0 - 7 sets is as x
2. Uses Iloc to get the last column sets it as y
3. Uses train_test_split to split the data into Training and test sets

In [164]:
```
x = diabetes.iloc[:, 0:8]
y = diabetes.iloc[:, 8] ## Print the last column using this method will not in
clude the column name on the top Payattention P1

    X_train, X_test, Y_train, Y_test = train_test_split(x,y, random_state = 0,
test_size = 0.2)
```

1. Standardized Scaler is used to normalize the data. This is particular important as without scaling the distance between the points on the vertical access may dominate or appear much larger than the distance in the horizontal access or vice versa. In methods like the KNN where distance is used to determine outcome, distance is vital. [https://www.youtube.com/watch?v=sxEqtjLC0aM (https://www.youtube.com/watch?v=sxEqtjLC0aM)](https://www.youtube.com/watch?v=sxEqtjLC0aM)

In [333]:
```
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

1. In the K Neearest neighbor Classifier, N_neighbors is the number of neightbors. It should be odd. One way to find the optimal number is find the square root of the length of X test. If it is even you subtract 1 to make it odd

In [334]:
```
math.sqrt(len(X_test))
```

Out[334]:
```
12.409673645990857
```

1. K Nearest Neighbor Object being used

In [335]:
```
KNN = KNeighborsClassifier(n_neighbors = 11, p = 2, metric ='euclidean')
KNN.fit(X_train, Y_train)## not including the column name on the top will allo
w the Y_train to be changed to a 1 dimensional
## array Payattention P2. The reason that this does not happen to the X train
despite it having columns is that
##in the above standard scaler and fit. transform funcntion x train is changed
to an array

y_pred = KNN.predict(X_test)
y_pred
```

Out[335]:
```
array([1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      dtype=int64)
```

1. Confusion Matrix to show accuracy, recall and precision

In [347]:
```
#cm = confusion_matrix(Y_test, y_pred)
#cm

#Labels = ['TP 94','FP 13','FN 16','TN 31']
#Labels = np.asarray(Labels).reshape(2,2)
#sns.heatmap(cm, annot = Labels, fmt = '', cmap='Blues')


confusion_matrix = metrics.confusion_matrix(Y_test, y_pred)
confusion_matrix
```

Out[347]:
```
array([[94, 13],
       [16, 31]], dtype=int64)
```

In [348]:
```
accuracyscore = accuracy_score(Y_test, y_pred)
precisionscore = precision_score(Y_test, y_pred)
recallscore = recall_score(Y_test, y_pred)
f1score = f1_score(Y_test, y_pred)

print(accuracyscore, recallscore, precisionscore,f1score)
```

```
0.8116883116883117 0.6595744680851063 0.7045454545454546 0.6813186813186813
```

In [354]:
```
from sklearn.metrics import classification_report

pd.DataFrame(classification_report(Y_test, y_pred, output_dict = True))
```

Out[354]:

	0	1	micro avg	macro avg	weighted avg
f1-score	0.866359	0.681319	0.811688	0.773839	0.809886
precision	0.854545	0.704545	0.811688	0.779545	0.808766
recall	0.878505	0.659574	0.811688	0.769040	0.811688
support	107.000000	47.000000	154.000000	154.000000	154.000000

If there are large class imbalances with the data with the minority less than 20%, it is then very beneficial to use other metrics besides accuracy to check the the model's validity. Since imbalances here are at 30% accuracy is sufficient

In [ ]: