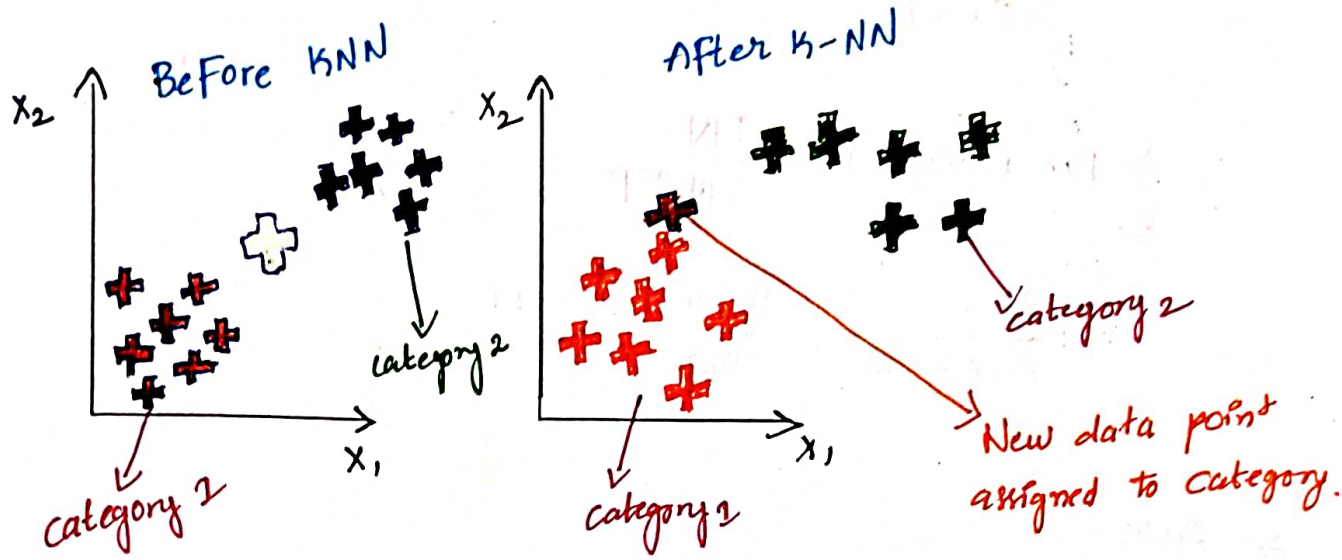
Dt:- 20/4/22
10:30pm

KNN [K-Nearest Neighbors]



Que:- How to chose KNN number ?

A: Step:1 Choose the number "K" of neighbours

Step:2 Take the K- nearest neighbours of new data point. according to Euclidean distance.

Step:3 Among these K- neighbors, Count the number of data points in each category

Step:4 Assign the new data point to the category where you counted most neighbours.

Ex:-

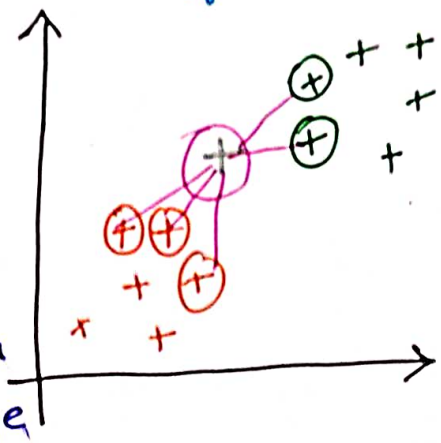
$K = 5$

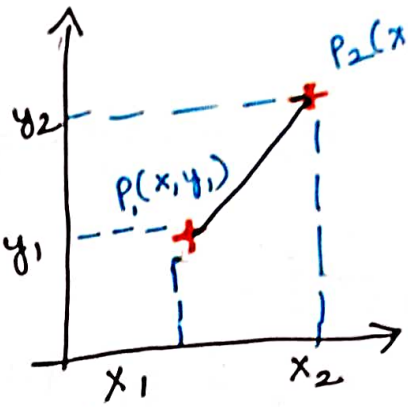
* orange = 3
* green = 2

$\therefore$ 3 is highest.

So, Convert data Point to orange

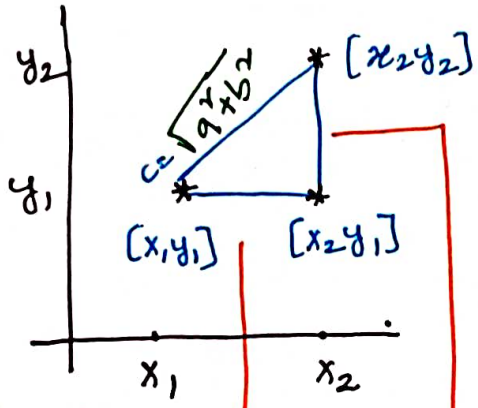
* If K = 4 [Even number]
* check sum of Distances of 1st category and sum of Distances of 2nd category which is having less distance select...



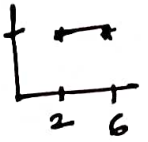


* Euclidean distance b/w P_1 & P_2 = $\sqrt{[x_2-x_1]^2+[y_2-y_1]^2}$

Distance Formula



Ex:-



(2,4)
(6,4)

distance $a = [x_2 - x_1]$

distance $b = [y_2 - y_1]$

Since, it is right angle Triangle.

"Hypotenuse" Theorm

$a^2 + b^2 = c^2$

$c = \sqrt{a^2+b^2}$

$c = \sqrt{a^2+b^2} = \sqrt{[x_2-x_1]^2+[y_2-y_1]^2}$

* Manhattan Distance :- $|[x_2-x_1]| + |[y_2-y_1]|$ → Distance can't be negative value.

* Euclidean Distance :- $\sqrt{[x_2-x_1]^2+[y_2-y_1]^2}$

Backend Code of "KNN"

```
def ebenhaezer_knn(x,y):
    d = []
    for i in data:
        distance =
        d.append(distance)
    d.sort()
    d[0.5]
```

name

out: 01234

CODE

Data Understanding

To understand KNN for classification, we'll work with a simple dataset representing gene expression Levels. Gene expression level are calculated by ratio between the expression of target gene (gene intrest) and expression of one or more reference genes (often household genes). this dataset is synthetic and specifically designed to show some of strengths and limitations of using KNN for classification

biopsy test cancer cells removing from body

load data

df = pd.read_csv("gene_expression_csv")

df.head()

Out:

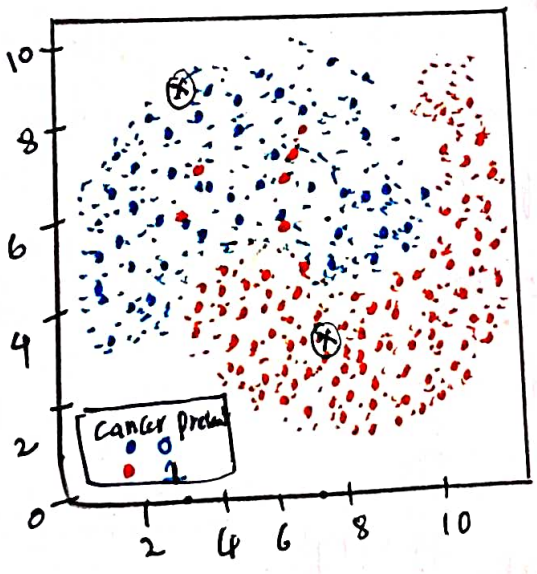
gene_One	gene_Two	Cancer_Present
4.3	3.9	1
2.5	6.3	0
5.7	3.9	1
6.1	6.2	0
7.4	3.4	1

Ex: 6.8, 3.9 → ? = 1
2.5, 8.2 → ? = 0

Exploratory Data analysis [EDA]

sns.scatterplot(x="Gene_one", y="Gene_two", hue="Cancer_Present", data = df).

Out:



plt.legend(loc=(1.1, 0.5))

Cancer Present
0
1

df.isnull().sum()

Out

Gene One	0
Gene two	0
cancer present	0

df.shape

Out: (3000, 3)

X & Y

```
# X = df.drop("Cancer Present", axis=1)
# y = df["Cancer Present"]
```

Train_test_split

```
from sklearn.model_selection import train_test_split
# x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=..., Random state = 29)
```

*** **Scaling** → For KNN [Mandatory] ⇒ $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$ ← [Distance]

```
From sklearn.preprocessing import StandardScaler
#  Scaler = StandardScaler
#  x_train = Scaler.fit_transform(x_train)
#  x_test = Scaler.fit_transform(x_test)
```

EX:- 1000 4
4000 5
9000 6

$\sqrt{(1000-4000)^2+(4-5)^2}$
$= \sqrt{(6000)^2+1^2}$
$= \sqrt{36000000+1}$

That's why scaling is Mandatory.

MODELLING

```
from sklearn.neighbors import KNeighborsClassifier
# knn_model = KNeighborsClassifier()
# knn_model.fit(x_train, y_train)
```

→ Help default

* K = 5
* p = 1 (manhattan distance)
* p = 2 [Euclidean distance]

Out! KNeighborsClassifier

Prediction

```
# y_Pred_test  = knn_model.predict(X_test)
# y_Pred_train = knn_model.predict(x_train)
```

Evaluation

```
from sklearn.metrics import accuracy_score

# accuracy_score(y_test, y_Pred_test)
# accuracy_score(y_train, y_Pred_train)
```

Out: 0.927 [Test]
0.943 [Train]

⇒ CV

```
from sklearn.model_selection import cross_val_score

# scores = cross_val_score(knn_model, x, y, cv=5)
# scores.mean()
# print(scores)
```

Out: [0.918, 0.93, 0.925, 0.92, 0.93]
0.926

Que: How to choose Best K-value ?

A:- By "Hyper parameter Tunning".

Nested Forloop

```
for K in range (1,31):
    for P in [1,2]:
        print (K)
        print (P)
```

K-NN [Hyper Parameter] Tunning

Backend code

```
from sklearn.model_selection import GridSearchCV

# estimator = KNeighborsClassifier ()

# param_grid = {"n_neighbors": list (range (1,31)), "p": [1,2]}

# full_cv_classifier = GridSearchCV (estimator, param_grid, cv=5,
                                     scoring = "accuracy")

# full_cv_classifier.fit (x_train, y_train)
```

out: GridSearchCV [cv=5, estimor = KNeighborsClassifier (),
param_grid = {"n_neighbors": [1,2,3,4,5,6,7,
8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,
23,24,25,26,27,28,29,30],
"p": [1,2]}, scoring = "accuracy")

```
# full_cv_classifier.best_params_
```

out: {'n_neighbors': 20, 'p': 2}

Train accuracy is only used for checking for overfitting (or) underfitting.

Final Model

→ with K = 20, P = 2

```
# knn = KNeighborsClassifier ( n_neighbors=29  P=2)
# knn.fit (x_train, y_train)
```

Out: KNeighborsClassifiers (n_neighbors = 20)

```
# Test
# y_pred = knn.predict (x_test)
# accuracy_score (y_test, y_pred)
```

Out: 0.946

```
from sklearn.metrics import confusion_matrix
# confusion_matrix ( y_test, y_pred)
```

Out: array ([[451 , 19],
[29 , 401]],

```
from sklearn.metrics import classification_report
# Print ( Classification_report (y_test, y_pred)
```

	Precision	recall	f1-Score	Support
0	0.94	0.96	0.95	470
1	0.95	0.93	0.94	430
accuracy			0.95	900
Macro Avg	0.95	0.95	0.95	900
Weighted Avg	0.95	0.95	0.95	900

Train. accuracy

```
# y_pred_train = knn.predict(x_train)
# accuracy_score(y_train, y_pred_train)
```

Out: 0.933

21/04/22
3:30AM