# KNN = Theory

KNN : It's the Simplest yet one of the most Commonly used M.L. algo.

## Advantages of KNN

→ Easy to implement

→ KNN has no-model - No complex function like other ML algorithms such as Linear Regression, Logistic Regression, Decision Tree, Naïve Bayes etc

→ It just stores data, so no learning is required.

→ Training is not required hence all work is done during prediction.

→ No prior knowledge or any knowledge about distribution of data required.

→ Defers data processing untill receives request to classify unlabled data - Lazy Learner.

# Example of application of KNN

KNN is simple but powerful algorithm & many times outperforms other algorithms.

## Is my heart functioning well?

→ In medical research lots of data are generated.

→ KNN is capable of handling huge amount of data provided we've sufficient resources,

→ And it doesn't get affected by outliers.

→ So based on similarity among cases of diagnosis it can predict diseases.

* Since it works on basis of similarity b/w instances so it has wide application in Recommender System.

→ Similarity b/w $\boxed{\text{users' ratings, product ratings}}$ can efficiently be $\boxed{\text{recognized}}$ by KNN algo.

N.B.:- KNN algo ~~[illegible]~~ will be talked & its application is discussed. in Recommender system.

## KNN in a nutshell

→ It's a form of $\boxed{\text{Supervised learning}}$

* Data processed by KNN has input features & output feature.

→ Used in both $\boxed{\text{Classification \& Regression}}$

→ Uses $\boxed{\text{nearest Majority}}$ of neighbors to predict data points.

Scanned by CamScanner

# How does KNN work?

* Step 1: Build a neighborhood with $\boxed{\text{K-members}}$
(K = 1 to any no, only $\boxed{\text{odd NO}}$ because the algo has to choose the heighest No of rankings)

* Step 2: Find $\boxed{\text{distance}}$ from $\boxed{\text{query point}}$ (Test data) to each point of neighborhood.

* Step 3: $\boxed{\text{Assign}}$ query point to that class having $\boxed{\text{maximum members around}}$ (majority voting).

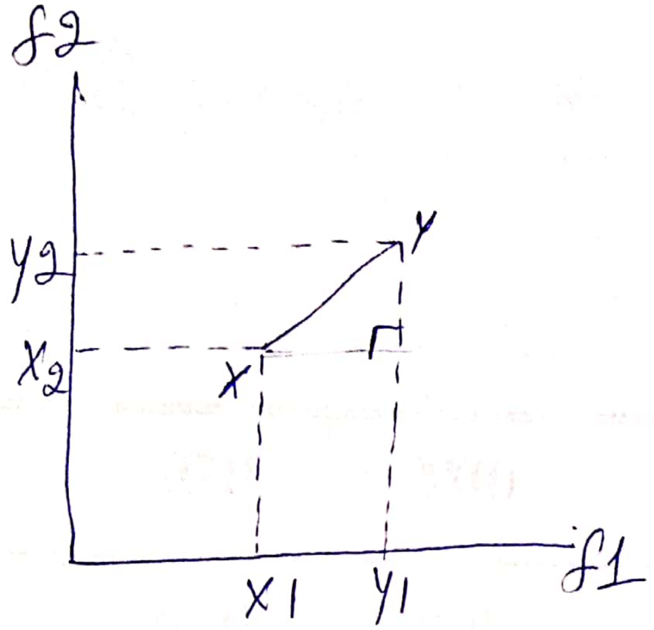
→ classification is done in KNN based on $\boxed{\text{majority of neighbors.}}$

→ where $\boxed{K}$ denotes number of neighbors.

---

There are a no of distance measures for continuous data such as Euclidean, manhattan, Minkowski; most common being $\boxed{\text{Euclidean Distance}}$

Euclidian distance measures shortest distance b/w two points (X & Y)




$$\boxed{\Rightarrow \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}}$$

$X(X1, X2)$ & $Y(Y1, Y2)$

$f1 = X1, Y1$. & $f2 = X2, Y2$

$$\boxed{d = \sqrt{(Y1 - X1)^2 + (Y2 - X2)^2}}$$

For Continuous Data (Regression)

Euclidian distance is used [as mentioned above]

For Categorical Data (Classification)

Hamming distance is used.

## Hamming distance is measured as below

This is locations where binary vectors (2 classes) differ.

$$\boxed{D_H = \sum_{i=1}^{k} |x_i - y_i|}$$

$$x = y \Rightarrow D = 0$$
$$x \neq y \Rightarrow D = 1$$

| X | Y | Distance |
|---|---|---|
| Male | male | 0 |
| Male | Female | 1 |

← Sequence of X & Y differs (Male & Female)

A B B A A B B A B A
B B A A A B A B B A

Hamming distance is 4, because classes differ at 4 locations.

# KNN

① Import packages (Load the data)
~~4~~ ③ Clean data — Missing values
outlier treatment
② EDA — Box plot
Correlation
Histogram
pairplot
many others

## Step-1

~~4~~ Create features and Labels

X = . . - - - - - -
y =
X.shape
y.shap .

If the data is in np. array Convert it into Pd. DataFrame

df = Pd.DataFrame(file).apply(Pd.to_numeric)

Step 2

## Splitting the data into Train & Test ~~data~~ Set

Need to verify

from sklearn.model_selection import train_test_split

Step 3 Create instant of the model

## Building the KNN Model

from sklearn.neighbors import KNeighborsClassifier

clf_knn = KNeighborsClassifier(n_neighbors=10)

Step 4 => Fit the model

clf_knn = clf_knn.fit(X_train, y_train)

Step 5 => Predict the model

y_pred = clf_knn.predict(X_test)

Step 6 => Evaluate the model

Cross validation is used to see how well a model performs in an independent dataset. Different samples are used for training & test & accuracy score is calculated

from sklearn.model_selection import cross_val_score
score_knn = cross_val_score (clf_knn, X_test, y_test, cv=5)
→ print ("Cross Validation score:" + str(score_knn))
print ("Cross Validation mean score:" + str(score_knn.mean()))

## Classification report

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score.

cr = classification_report(y_test, y_pred)
pr(cr)

## Confusion matrix

cm = confusion_matrix (y_test, y_pred).
cm

## Optimization of Neighbours

↳ Use different counts of neighbours by setting value of k (5, 7, 9, 11, 13, 15) => Any range of values
↳ Perform CV(5) on each value of k (k in KNN)
↳ Compare the CV score to deduce the best model.
↳ Use the best value of k.

Scanned by CamScanner

Grid search

list_k = [5,7,9,11,13,15] => Always odd No / Even No is also fine

```
for k in list_k:
    Clf_knn = KNeighboursClassifier(n_neighbors = k)
    score_knn = Cross_val_score(Clf_knn, X_test,
                                y_test, cv=5)
    Print ("k: ", k)
    Print ("Cross validation score: " + Str(score_knn))
    Print ("Cross validation mean score: " + Str(score_knn.mean()))
    print (" ")
```

~~we can~~ We can use Grid search to get faster result.