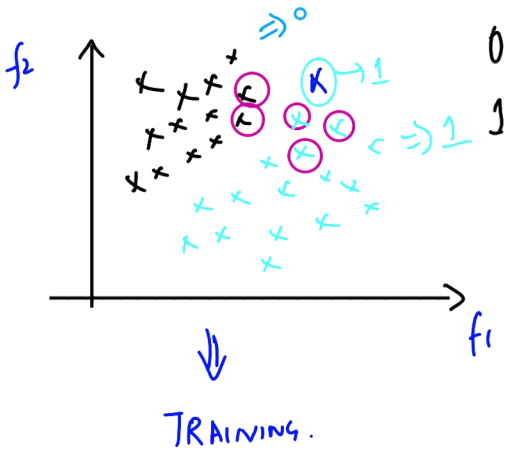
K Nearest Neighbour (KNN)

1. Classification
2. Regression

1. Classification

K=5



f1	f2	y [Binary Categories]
—	—	0
		1

TRAINING.

1. We have to initialize the K value — K=5

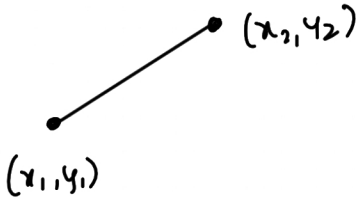
   K > 0 ... ∞

   K = 1, 2, 3, 4, 5 ..... ⇒ Hyperparameter

2. Find the K Nearest Neighbour for The Test Data.

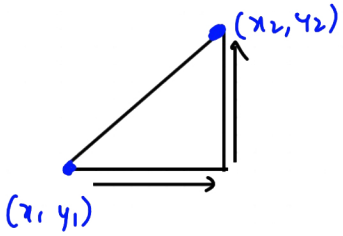
3. From those K=5 how many neighbour belong to 0 Category and 1 Category

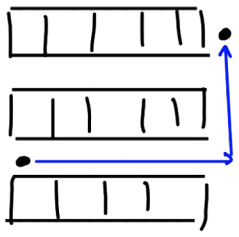
1. Euclidian Distance



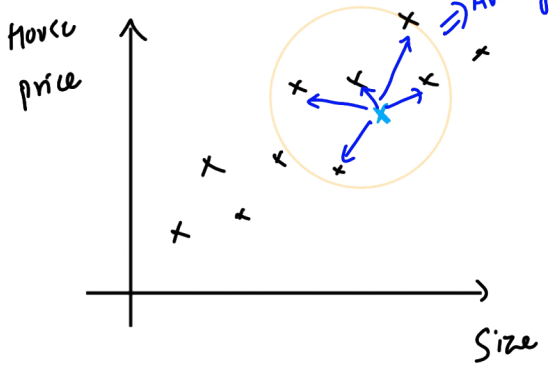
$$\text{Distance} = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

2. Manhattan Distance



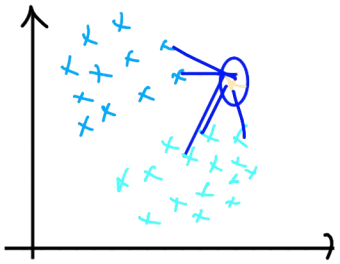


② Regression



Median

⇒ Average of all these points to find out the o/p



Time complexity

O(n) ⇒ Million of datapoint

{ ① KD Tree
② Ball Tree } ⇒ Optimize

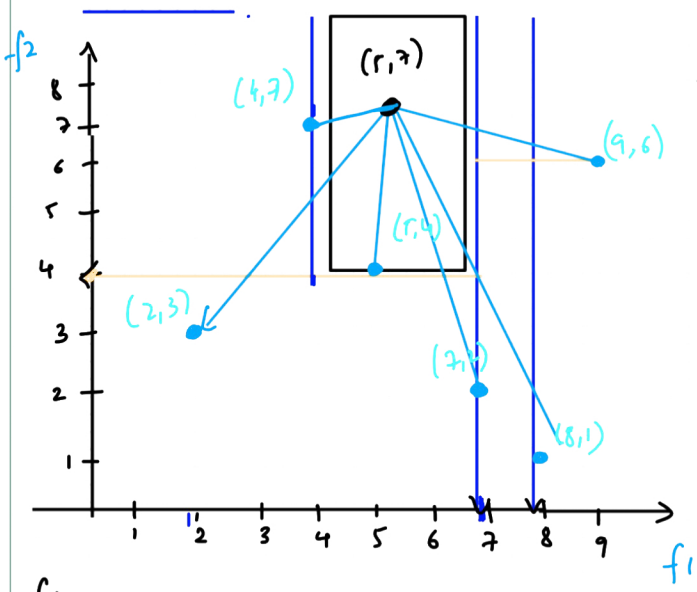
⇓

Binary Tree

so, time complexity is n^{th} time of KNN
we use KD tree or Ball Tree because they use Binary tree(optimized)

Variants of KNN

KD Tree



f1

2, 4, [5, 7] 8, 9

$\frac{5+7}{2} = \frac{12}{2} = 6.5$

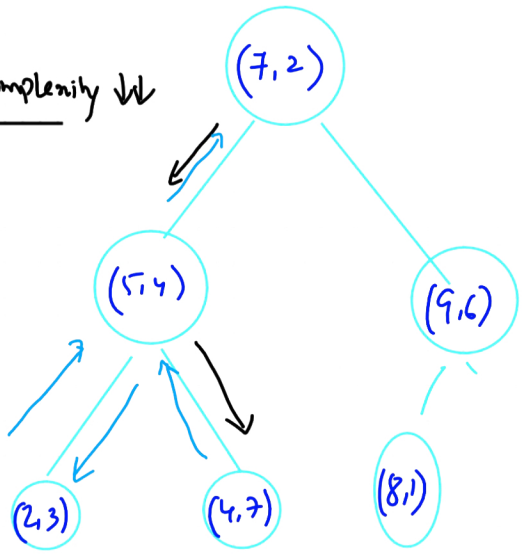
f2

1, 2, [3, 4], 6, 7

f1	f2
7	2
5	4
9	6
2	3
4	7
8	1
Median	Median

Binary Tree [KD Tree]

Time Complexity ↓↓



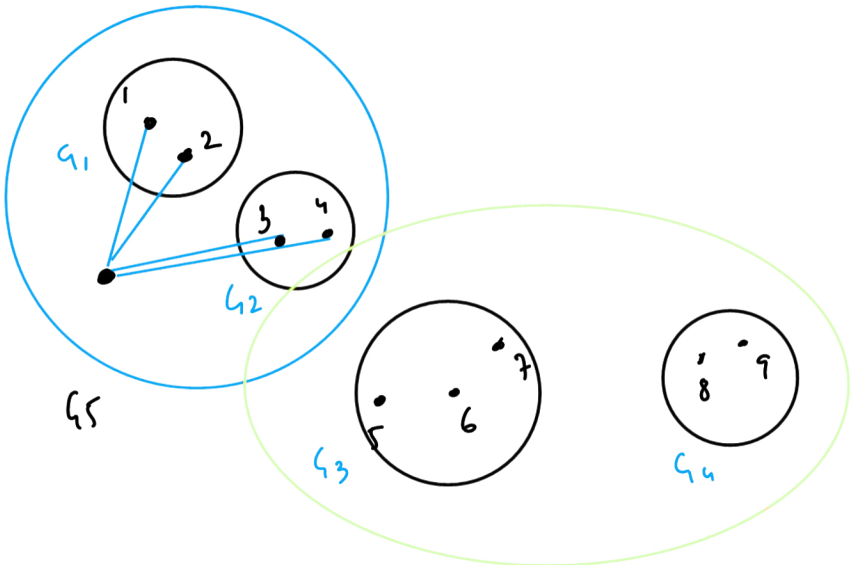
Back Tracking

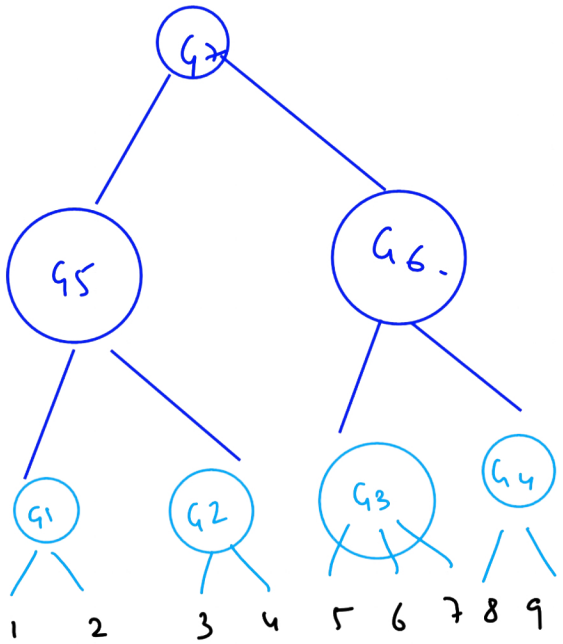
it splits dataset by median of features one-by-one and create binary tree

so, for new data it just look for nearest datapoints around it and calculate average of them for prediction

② Ball Tree

Time Complexity ↓↓





create circle or closed groups around near datapoints and then creates binary tree

Machine_Learning_K-Nearest_Neighbors_KNN_KDtree_BallTree

1 K Nearest Neighbour Classifier

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

[2]:
```
from sklearn.datasets import make_classification

X, y =make_classification(
    n_samples=1000, # 1000 observations
    n_features=3, # 3 total features
    n_redundant=1,
    n_classes=2, #binart target/label
    random_state=999
)
```

[3]:
```
X
```

[3]:
```
array([[-0.33504974,  0.02852654,  1.16193084],
       [-1.37746253, -0.4058213 ,  0.44359618],
       [-1.04520026, -0.72334759, -3.10470423],
       …,
       [-0.75602574, -0.51816111, -2.20382324],
       [ 0.56066316, -0.07335845, -2.15660348],
       [-1.87521902, -1.11380394, -4.04620773]])
```

[4]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
```

[5]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[6]:
```
classifier = KNeighborsClassifier(n_neighbors=5, algorithm='auto')
## algorithm we can select kd_tree, ball_tree or brute search, for now we are␣
 ↪selecting auto
#auto' will attempt to decide the most appropriate algorithm based on the␣
 ↪values passed to fit method.
## by hyperparameter tuning(random or grid searchCV) we can get all optimal␣
 ↪parameters
classifier.fit(X_train, y_train)
```

[6]:
```
KNeighborsClassifier()
```

by-default p='2' means euclidean distance not '1'-> manhattan distance - whenever we have features that are randomly distributed in feature dimensions, we should always take euclidean distance

[7]:
```
y_pred = classifier.predict(X_test)
```

[8]:
```
from sklearn.metrics import confusion_matrix, accuracy_score,␣
 ↪classification_report
```

[9]:
```
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[158  11]
 [ 20 141]]
0.906060606060606
              precision    recall  f1-score   support

           0       0.89      0.93      0.91       169
           1       0.93      0.88      0.90       161

    accuracy                           0.91       330
   macro avg       0.91      0.91      0.91       330
weighted avg       0.91      0.91      0.91       330
```

[ ]:
```

```

Task

GridseachCV for i k = 1,2,3,4,5,6,7,8,9,10

K best

[10]:
```
from sklearn.model_selection import GridSearchCV
```

[11]:
```
param_grid = {'n_neighbors':
              [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
             }
```

[22]:
```
clf = GridSearchCV(classifier,param_grid=param_grid)
```

[23]:
```
clf.fit(X_train,y_train)
```

[23]:
```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]})
```

[12]:
```
clf = GridSearchCV(classifier,param_grid=param_grid, scoring='accuracy',␣
↪verbose=2) #cv=5 by-default
```

[13]:
```
clf.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
[CV] END …n_neighbors=5; total time=   0.0s
[CV] END …n_neighbors=5; total time=   0.0s
[CV] END …n_neighbors=5; total time=   0.0s
[CV] END …n_neighbors=5; total time=   0.0s
[CV] END …n_neighbors=5; total time=   0.0s
[CV] END …n_neighbors=6; total time=   0.0s
[CV] END …n_neighbors=6; total time=   0.0s
[CV] END …n_neighbors=6; total time=   0.0s
[CV] END …n_neighbors=6; total time=   0.0s
[CV] END …n_neighbors=6; total time=   0.0s
```

```
[CV] END …n_neighbors=7; total time=   0.0s
[CV] END …n_neighbors=7; total time=   0.0s
[CV] END …n_neighbors=7; total time=   0.0s
[CV] END …n_neighbors=7; total time=   0.0s
[CV] END …n_neighbors=7; total time=   0.0s
[CV] END …n_neighbors=8; total time=   0.0s
[CV] END …n_neighbors=8; total time=   0.0s
[CV] END …n_neighbors=8; total time=   0.0s
[CV] END …n_neighbors=8; total time=   0.0s
[CV] END …n_neighbors=8; total time=   0.0s
[CV] END …n_neighbors=9; total time=   0.0s
[CV] END …n_neighbors=9; total time=   0.0s
[CV] END …n_neighbors=9; total time=   0.0s
[CV] END …n_neighbors=9; total time=   0.0s
[CV] END …n_neighbors=9; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
```

[13]:
```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]},
             scoring='accuracy', verbose=2)
```

[14]:
```
clf.best_params_
```

[14]:
```
{'n_neighbors': 9}
```

[15]:
```
clf.best_score_
```

[15]:
```
0.9029850746268657
```

[17]:
```
clf.score ## just checking
```

[17]:
```
<bound method BaseSearchCV.score of
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]},
             scoring='accuracy', verbose=2)>
```

[18]:
```
y_pred = clf.predict(X_test)
```

[19]:
```
accuracy_score(y_test,y_pred)
```

[19]:
```
0.9121212121212121
```

[ ]:
```

```

[33]:
```
## with cross-validation cv=10
```

[20]:
```
clf = GridSearchCV(classifier,param_grid=param_grid, cv=10, scoring='accuracy',␣
↪verbose=2)
```

[21]:
```
clf.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=1; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=2; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=3; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
[CV] END …n_neighbors=4; total time=   0.0s
```

```
[CV] END …n_neighbors=9; total time=   0.0s
[CV] END …n_neighbors=9; total time=   0.0s
[CV] END …n_neighbors=9; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
[CV] END …n_neighbors=10; total time=   0.0s
```

[21]:
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]},
             scoring='accuracy', verbose=2)
```

[22]:
```
clf.best_params_
```

[22]:
```
{'n_neighbors': 7}
```

[23]:
```
clf.best_score_
```

[23]:
```
0.9014925373134328
```

[24]:
```
y_pred = clf.predict(X_test)
accuracy_score(y_test,y_pred)
```

[24]:
```
0.9151515151515152
```

[ ]:

[ ]:

1.1 KNN Regressor

[1]:
```
from sklearn.datasets import make_regression
X, y = make_regression(n_samples=1000, n_features=2, noise=10, random_state=42)
```

[2]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
```

[3]:
```
from sklearn.neighbors import KNeighborsRegressor
```

[5]:
```
regressor = KNeighborsRegressor(n_neighbors=6, algorithm='auto')
regressor.fit(X_train,y_train)
```

[5]:
```
KNeighborsRegressor(n_neighbors=6)
```

[6]:
```
y_pred = regressor.predict(X_test)
```

[7]:
```
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
```

[8]:
```
print(r2_score(y_test,y_pred))
print(mean_absolute_error(y_test,y_pred))
print(mean_squared_error(y_test,y_pred))
```

```
0.9189275159979495
9.009462452972217
127.45860414317289
```

[ ]: