24 feb 23
Friday

K-Nearest Neighbours (KNN)

↳ Type: Supervised Learning Algorithm → finding relationship b/w I/P & O/P
Task: Classification & Regression

Eg Problem Statement → Given a datapoint of Height & Weight, predict the Gender.

(H, W) → ML model → Gender
I/P   O/P

H	W	G
150	45	F
180	80	M
145	50	F
190	75	M

Input → H, W
Output → G

Classification or Regression

1. Distance based Approach → KNN
2. Boundary " " → linear Reg (Reg), logistic Reg (class), SVM (both)
3. Rule " " → Decision Tree
4. Probabilistic " " → Naive Bayes (class)
5. Ensemble " "
6. Deep learning " "

Q Is the data available? → yes
Q Does the data have 'y'? → yes
Now Algo which can be used

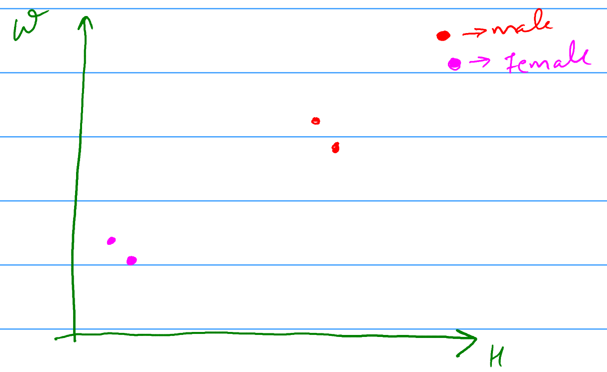
Evaluation Metrics
↳ Accuracy, precision, F1 score, confusion matrix. etc.

H	W	G
150	45	F
180	80	M
145	50	F
190	75	M

Collection of rows / datapoints

Dataframe → matrix

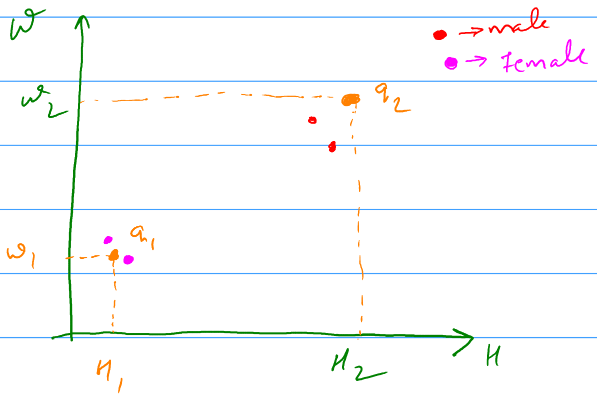
Visualizatⁿ →



Now, given a query point, predict the gender
(H, W)

H	W	G
150	45	F
180	80	M
145	50	F
190	75	M

Visualizatⁿ →



$q_1 \rightarrow (H_1, W_1) \longrightarrow$ Female

$q_2 \rightarrow (H_2, W_2) \rightarrow$ Male

Basic KNN steps

Given a query point
- ↳ find its nearest neighbour.
- ↳ Check the target values of the nearest neighbours.
  - ↳ (Count the no. of male & females)
- ↳ Majority class wins and query point becomes that class

Given a query point & the historical data

pre a Decide the value of K ← Done using hyperparameter tuning.

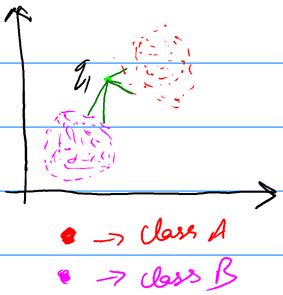
↳ K decides how many nearest neighbours to look at.

* Sklearn by default takes k=5

K=3 [odd] ↓ only for binary classification
doesn't work for multi class classificatⁿ

~~B~~ (a) Find the nearest neighbours

↳ K nearest Neighbours.



(b) Voting (Counting)

↳ Count the nearest neighbours belonging to each class.

(c) Majority Vote wins → decide the class of query point based on majority voting

Algorithm (K-NN) ↴

Steps:

1. Find the K-nearest neighbour.
2. Voting
3. Majority Voting

① Decide K
↳ no. of nearest values / nearest neighbours.

② distances = [ ]
↳ To store all the distances of query point from data set points.

③ KNNpts = [ ]

finding K neighbours

```
④ foreach datapoint in distances:
        ↳ Compute the distance b/w querypoint & datapoint.
                          ↳ Euclidean (by default, but can be changed as well)
        → dist = Distance (querypoint, datapoint).

        → distances.append((datapoint, dist))

⇒ distances = [(dP1, 100), (dP2, 5), -----]

⑤ Sort distances:
        distances = [(dP2, 5), (dp5, 6) ----]

⑥ KNN-pts = distances [ : K]

⑦ Voting:   count_male = 0
            count_female = 0
            for pt in KNN-pts:
                ↳ if pt is male:
                        count male += 1
                  else
                        count_female += 1
```

Voting

(8)

```
If count-male > count-female > 0
        print('male')
Else
        print('female')
```

The reason for KNN having such a fast training time is because it just memorizes the data during training time and there is no learning. All the work happens when a query point comes thats when it calculates the dist., sorts them and finds K nearest neighbours, as a result it has a large prediction time

Q What is the range of K?

K → 1 to n

Q Why not even K?

↳ In case of binary classification, there can be a situation of equal voting as a result we fail to classify the query point

↳ In case of multi-class classification, the K value doesn't matter as in either case we can get equal vote situation.

A → 0
B → 2
C → 2 } x?

A → 1
B → 1
C → 1 } x?

Q What happens if 'k' is too small or too large?

Range of K = 1 to n

K=1 → Can cause overfitting

K=n → Can cause underfitting

Q How to decide the value of K?

↳ Hyperparameter tuning.

It is a (hyperparameter of KNN) → There is dist. metric also

X_train
y_train } → KNN → model

Overfit

Underfit

Best fit

☆ KNN is a non-parametric ML algorithm

☆ KNN is aka Lazy Learner.

☆ KNN has a very high time & space complexity.

↳ KNN is not used in production.

☆ Best Alternative for KNN

{ ↳ ANN (Approximate Nearest Neighbours)

↳ K-d tree Data Structure }

↳ Improve the efficiency of KNN

☆ for KNN → Training Phase → very less time, high space complexity

aka lazy learner ⇐ no training at all

→ Testing Phase → almost all steps happen here ⟹ high prediction time

for each query point all dist. in df are calculated + Sorting is very time consuming

Selection Crieteria for Product^n ML model

- ↳ Model train time
- ↳ model prediction time
- ↳ Model Size

eg

① $X\_train, y\_train \rightarrow \boxed{\text{KNN}} \rightarrow M_1$ (Model)

$X\_test \rightarrow M_1 \rightarrow y\_test\_pred$

$y\_test, y\_test\_pred \rightarrow \boxed{\text{Eval}} \rightarrow \text{Score}$

② $X\_train, y\_train \rightarrow \boxed{\text{logistic Reg}} \rightarrow M_2$

$X\_test \rightarrow M_2 \rightarrow y\_test\_pred$

$y\_test, y\_test\_pred \rightarrow \boxed{\text{Eval}} \rightarrow \text{Score}$

★ Now for the same data both the models give a score of 98%. then how to choose?

	M_1	M_2
Train time	0.000001 sec	5 hr
Test time	1 hr.	0.000001 sec
Size	100 GB	1 GB
cost per pred.	Rs 1	Rs 1000

Now to choose a model you need to understand the targeted user base behaviour:

- ↳ A daily internet user would expect faster output and the model would have to be light weight. ⇒ M2 will be chosen
- ↳ If the user base being targeted don't care about the execution time but want less cost per prediction
  ⇒ M1 will be chosen.

# Whenever ML models are deployed, there will always be tradeoffs but we need to select the model based on tradeoffs that prove to be the best fit/ benificial to us.

27 Feb 2023
Monday

KNN Regression

Type → Supervised Learning
Task → Regression

$$D_n = \{(x_i, y_i) \mid x_i \in R^{d-1}, y_i \in R\}$$

Classification

Problem Statement: Given the height & weight of an individual predict the gender of the individual

Target(y) → Gender

Input → Height & Weight

Data →

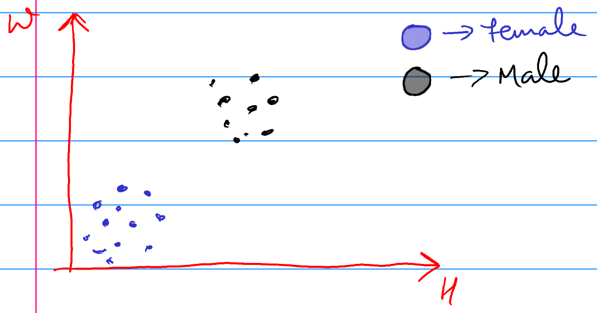
Height	Weight	Gender

$n \times d$

X | y

$$D_n = \{(x_i, y_i) \mid x_i \in R^{d-1}, \underbrace{y_i \in \{G, M\}}\}$$

Target → Discrete
⇓
Classification



Steps:

ⓐ decide the value of K.

ⓑ Compute the distance of x_q from each datapoint in the dataset

ⓒ Get K- nearest neighbours → Sort & slice

Regression

Problem Statement: Given the height of an individual, predict the weight of an individual.

Target (y) → Weight

Input → Height

Data →

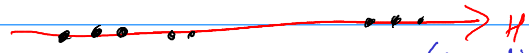
Height	Weight

$n \times d$

X | y

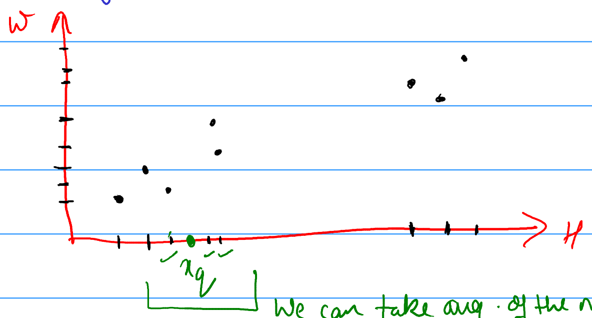
$$D_n = \{(x_i, y_i) \mid x_i \in R^{d-1}, \underbrace{y_i \in R}\}$$

Target → Real Value / Continuous Value
⇓
Regression



(One dimensional Representation)

Now, we try to visualize it in 2 dimensions



We can take avg. of the nearest neighbors and come to an aggregate value.

(d) Voting → Count the number of datapoints for each class. [Take the mode of the classes]

(e) Class with majority wins.

Basically computing the mode

In the above example, we find the 3 nearest neighbor and take the aggregate value (Avg. Value).

Here we can use two ways to compute the target,

mean [affected by outlier]

median [sorting will take O(nlogn) and O(n^2) worst case & lot of memory]

Steps:

(a) Decide the value of K.

(b) Compute the distance of x_q from each datapoint in the dataset.

(c) Get the K nearest neighbors → Sort & Slice.

(d) Compute the avg. from K-nearest neighbors.

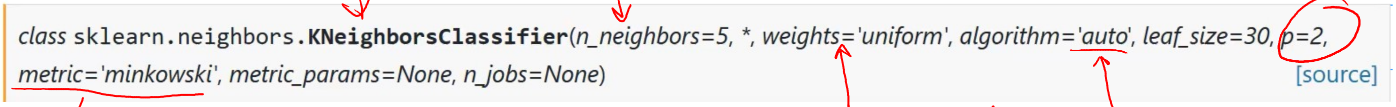
↳ Basically computing the mean/median.

When implementing KNN in sklearn we need to specify → K

→ Distance to be used [Euclidean, manhattan, minkowski etc.]

Parameters Same for Regressor also.

Value of K



class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None) [source]

for P=2, it by default computes Euclidean distances

assign importance to points by default everyone is equal.

Kdtree & other optimizing algo.

Hyperparameters for KNN

Their best value can be found using hyperparameter tuning

① value of K

② Distance metric by selecting the value of 'p'.

Alternate to KNN

→ ANN (Approximate Nearest Neighbor)

→ K-d trees.