MIS4311
Machine Learning Applications

Fall 2024

Lecture #6

Nearest Neighbors

Nearest neighbors provides functionality for **unsupervised and supervised neighbors-based learning methods.**

Unsupervised nearest neighbors is the foundation of many other learning methods.

Supervised neighbors-based learning comes in two flavors:

- **classification** for data with **discrete** labels
- **regression** for data with **continuous** labels

The principle behind nearest neighbor methods is to find a predefined number of **training samples closest in distance to the new point, and predict the label from this information**.
The number of samples can be a user-defined constant (**k-nearest neighbor** learning), or vary based on the local density of points (**radius-based neighbor** learning)

K-Nearest Neighbors

K-nearest neighbors (KNN) is a type of **supervised learning algorithm used for both regression and classification.**

KNN tries to **predict the correct class** for the test data **by calculating the distance between the test data and all the training points**.

Then select the K number of points which is closet to the test data.

The KNN algorithm **calculates the probability of the test data belonging to the classes of 'K'** training data and class holds the highest probability will be selected.

In the case of regression, the value is the **mean of the 'K' selected training points.**

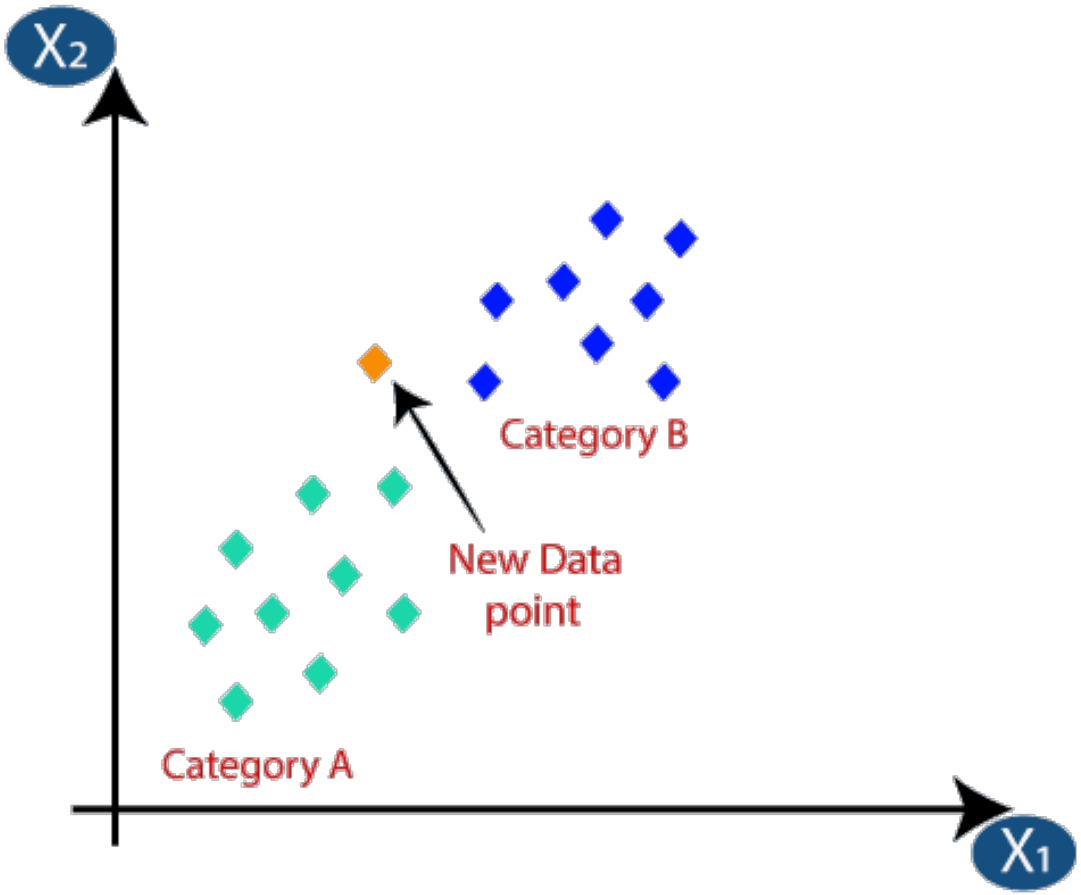
K-Nearest Neighbors

The K-NN algorithm:

- **Select the number K** of the neighbors
- **Calculate the Euclidean distance** of K number of neighbors
- **Take the K nearest neighbors** as per the calculated Euclidean distance
- Among these K neighbors, **count the number of the data points in each category.**
- **Assign the new data points** to that category for which the number of the neighbor is maximum.

K-Nearest Neighbors

Example: Suppose we have a new data point and we need to put it in the required category.

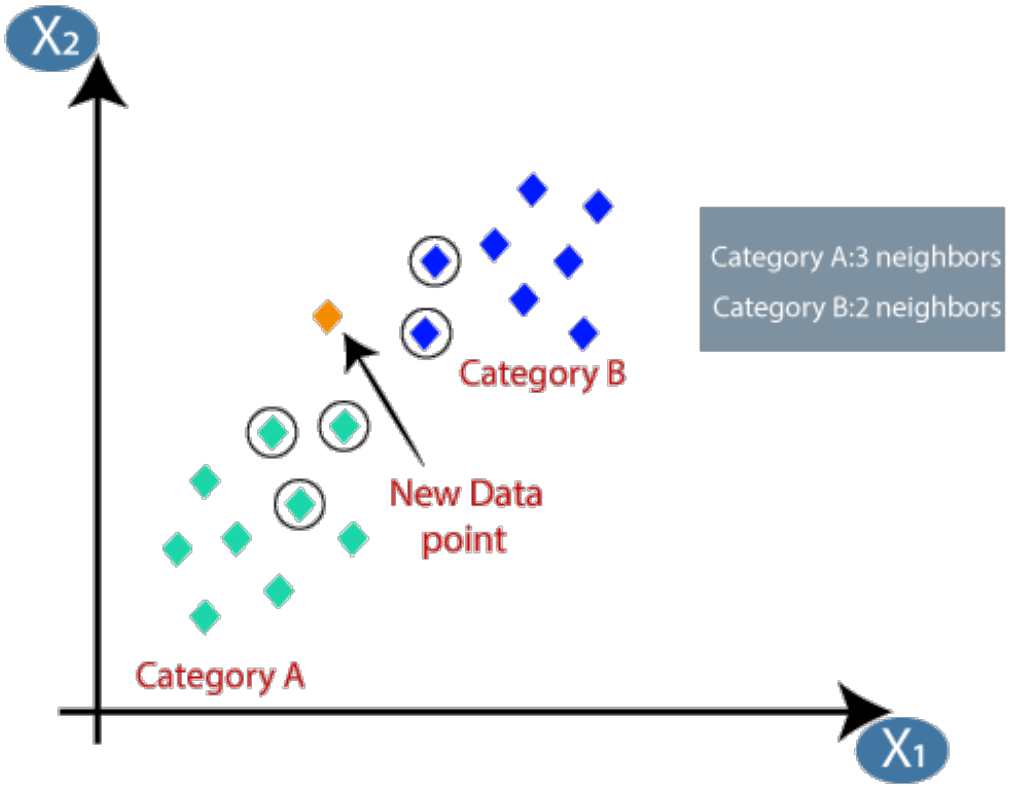


K-Nearest Neighbors

•Firstly, we will choose the number of neighbors, so we will choose the **k=5**.

•Next, we will **calculate the Euclidean distance** between the data points.

As we can see the **3 nearest neighbors are from category A**, hence this new data point must belong to category A.

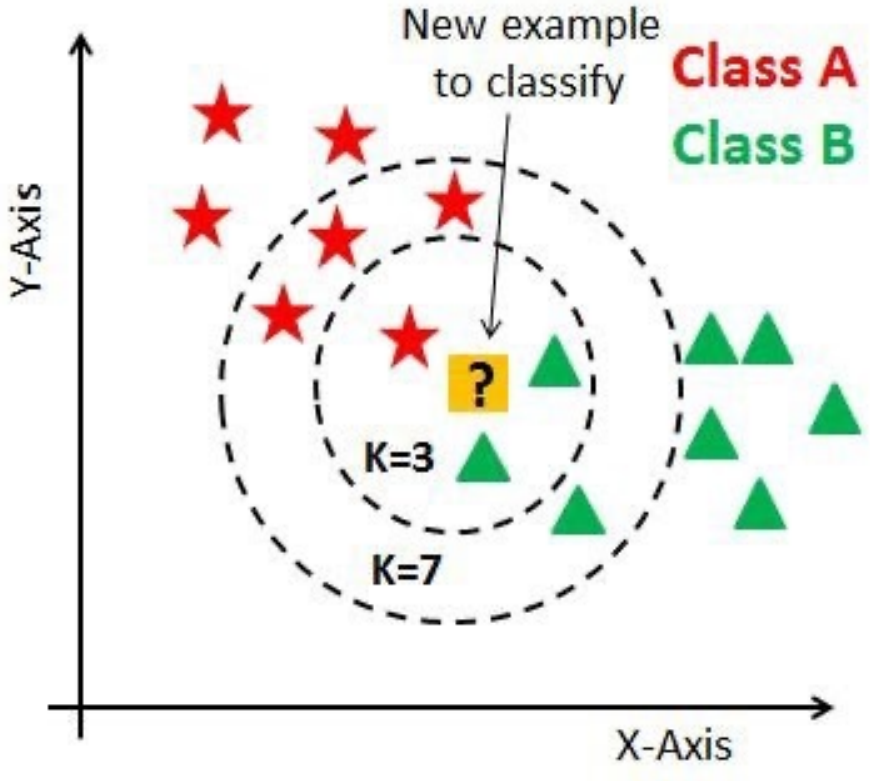


K-Nearest Neighbors

Selection of K value: K value indicates **the count of the nearest neighbors.** Updating distance metrics with **every iteration is computationally expensive,** and that's why KNN is a lazy learning algorithm.

If we proceed **with K=3,** then we predict that **test input belongs to class B**, and if we continue **with K=7,** then we predict that test input **belongs to class A.**

Thus the **K value has a powerful effect on KNN performance.**



K-Nearest Neighbors

Selection of K value: There are no pre-defined statistical methods to find the most favorable value of K.

•Initialize a random K value and start computing.

•Choosing a **small value of K leads to unstable decision boundaries.**

•The substantial **(larger) K value is better for classification as it leads to smoothening the decision boundaries**.

•Derive a **plot between error rate and K** denoting values in a defined range. Then **choose the K value as having a minimum error rate.**

K-Nearest Neighbors

Calculating distance: There are various methods for calculating this distance, of which the most commonly known methods are **Euclidian, Manhattan** (**for continuous**) and **Hamming distance (for categorical).**

Distance functions

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

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

Hamming Distance: It is **used for categorical variables**. If the value (x) and the value (y) are the same, the distance D will be equal to 0 . Otherwise D=1.

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

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

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

K-Nearest Neighbors

KNN classifier algorithms:

Brute Force:
Calculate the Euclidean distance from point of interest (of whose class we need to determine) to **all the points in training set**. Then we take class with majority points.

K-Dimensional Tree (KD tree):
KD tree is a hierarchical binary tree. It rearranges the whole dataset in a binary tree structure, so that when test data is provided, it would give out the result by traversing through the tree, which **takes less time than brute search.**

Ball Tree:
Similar to KD trees, Ball trees are also hierarchical data structure. These are very **efficient specially in case of higher dimensions**.

Nearest Neighbors - Regression

Neighbors-based regression can be used in cases where **the data labels are continuous** rather than discrete variables.

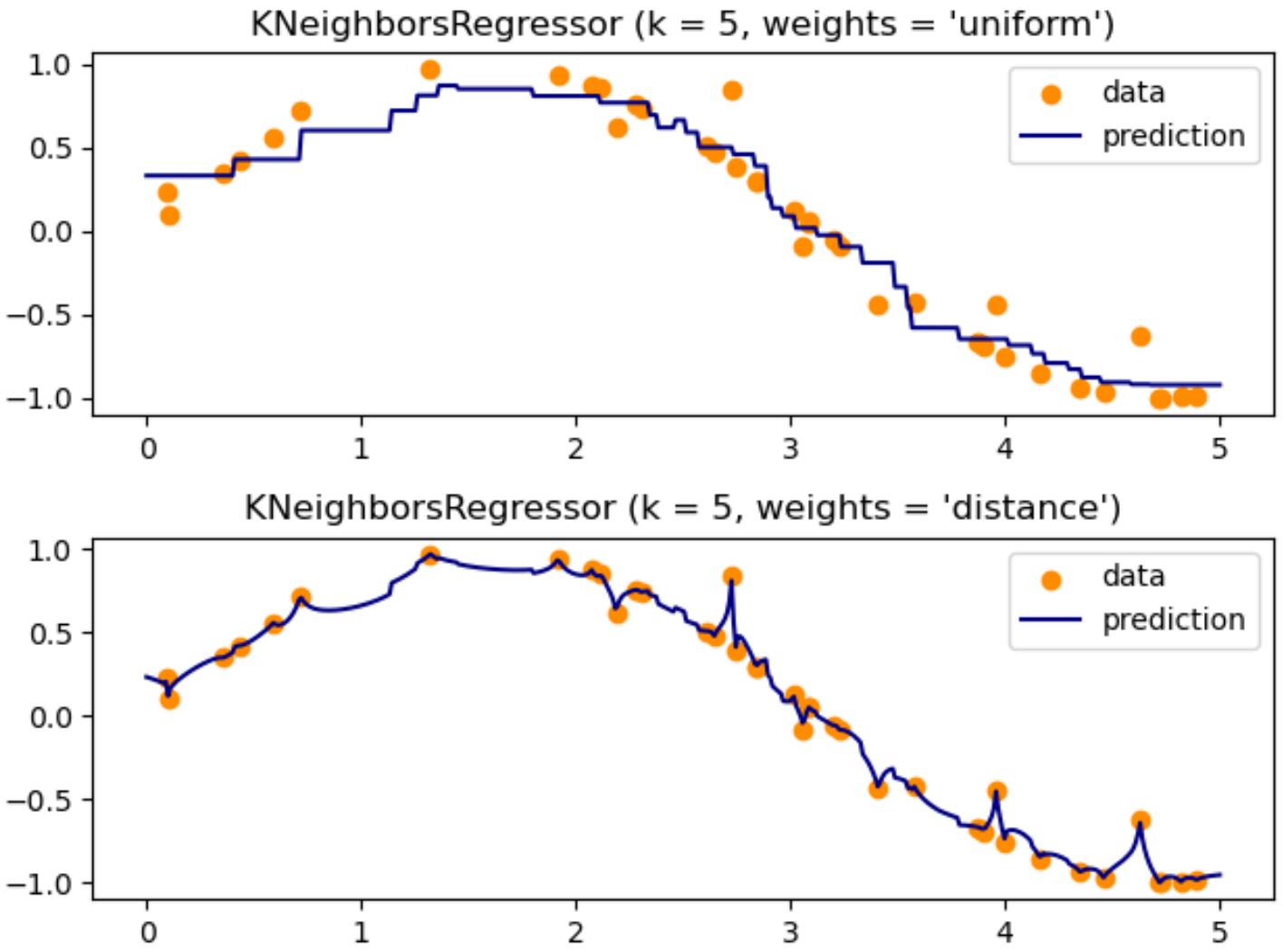
The label assigned to a query point is computed **based on the mean of the labels of its nearest neighbors.**

The basic nearest neighbors regression uses **uniform weights**.

The default value, **weights = 'uniform',** assigns equal weights to all points.

weights = 'distance' assigns weights proportional to the inverse of the distance from the query point.

Nearest Neighbors - Regression



K-means Clustering

K-means is an **unsupervised learning** method for clustering data points.

The algorithm **iteratively divides data points into K clusters by minimizing the variance in each cluster.**

Unsupervised learning is a conceptually different problem to supervised learning. Obviously, we **can't** hope to **perform regression**: we don't know the outputs for any points.

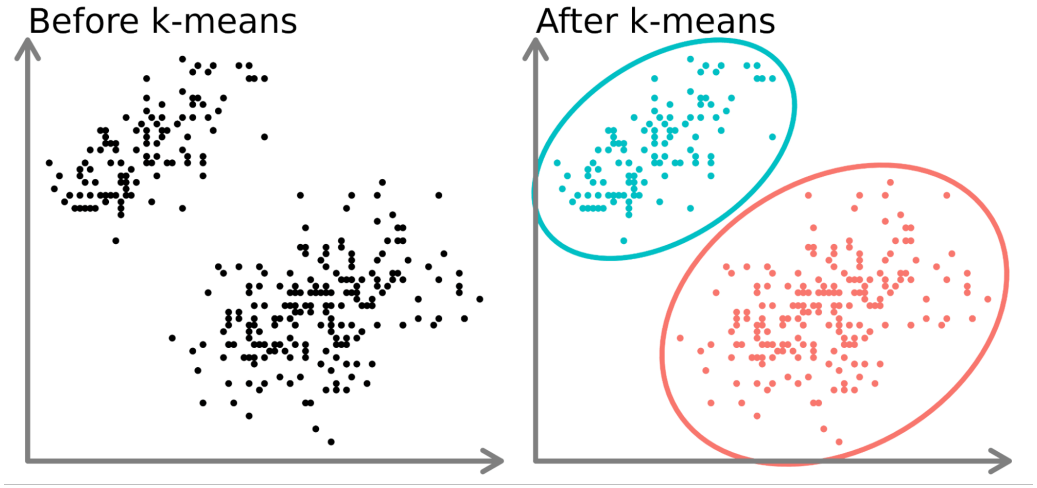
The aim of unsupervised learning is **to find clusters of similar inputs in the data** without being explicitly told that these datapoints belong to one class and those to a different class.

The algorithm has to **discover the similarities for itself.**

K-means Clustering

Suppose that we want to **divide our input data into K categories**, where we know the value of K (for example, we have a set of medical test results from lots of people for two diseases, and we want to see how well the tests identify the two diseases)

We **allocate K cluster centres to our input space**, and we would like to position these centres so that there is one cluster centre in the middle of each cluster.



K-means Clustering

How do we define the middle of a set of points? There are actually two things that we need to define:

A distance measure: In order to talk about distances between points, we need some way to measure distances. It is often the normal Euclidean distance

$$d((x,y),(a,b)) = \sqrt{(x-a)^2 + (y-b)^2}$$

The mean average Once we have a distance measure, we can compute the central point of a set of datapoints, which is the mean average.

A suitable way of positioning the cluster centres: we **compute the mean point of each cluster**, $\boldsymbol{\mu_c(i)}$, and put the cluster centre there.

This is equivalent to minimizing the **Euclidean distance** (which is the sum-of-squares error again) from each datapoint to its cluster centre.

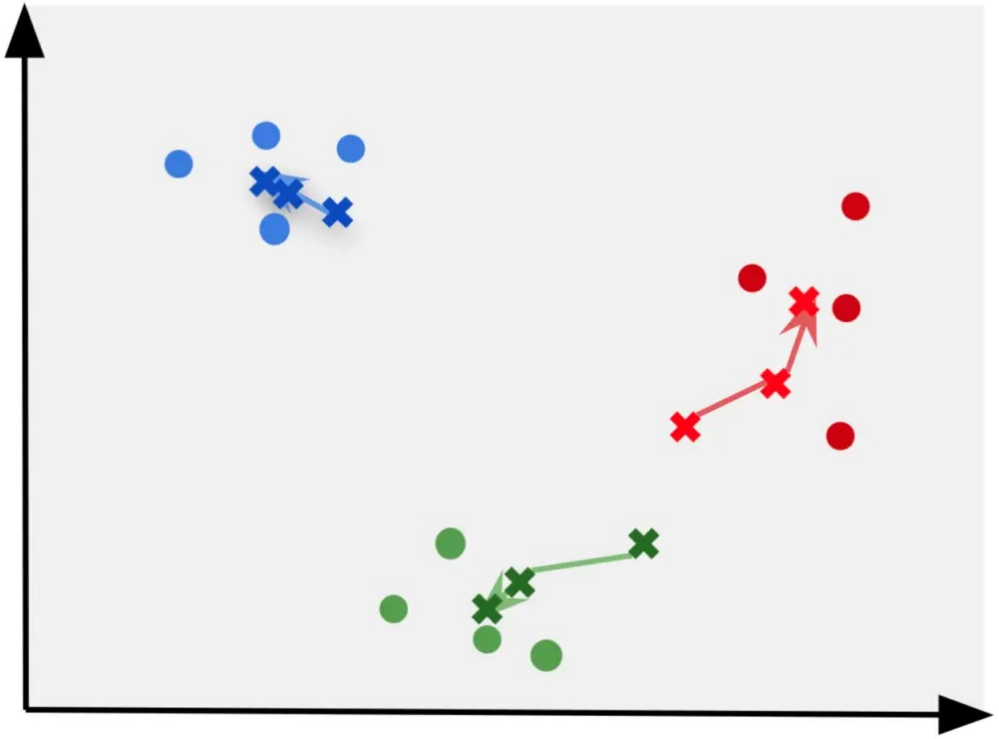
K-means Clustering

We start by **positioning the cluster centres randomly** through the input space, since we don't know where to put them, and then we **update their positions according to the data**.

We **decide which cluster each datapoint belongs to** by computing the distance between each datapoint and all of the cluster centres, and **assigning it to the cluster that is the closest**.

For **all of the points that are assigned to a cluster**, we then **compute the mean of them**, and **move the cluster centre** to that place.

We iterate the algorithm until the cluster centres stop moving.



K-means Clustering

No-math algorithm of k-means clustering:

1. Select the number of clusters (K)
2. Randomly select a number of data points that matches the number of clusters
3. Measure the distances between each point to its initial cluster
4. Assign each datapoint to its nearest initial cluster
5. Repeat the calculations for each point
6. Calculate the mean of each cluster
7. Assign the mean as the new cluster centroid
8. Measure each point to the new cluster centroid
9. Redefine clusters and assign the new mean as the next cluster centroid
10. Repeat process until convergence

K-means Clustering

- **Initialisation**
  - choose a value for k
  - choose k random positions in the input space
  - assign the cluster centres $\boldsymbol{\mu}_j$ to those positions
- **Learning**
  - repeat
    - for each datapoint $\mathbf{x}_i$:
      - compute the distance to each cluster centre
      - assign the datapoint to the nearest cluster centre with distance

$$d_i = \min_j d(\mathbf{x}_i, \boldsymbol{\mu}_j). \tag{14.1}$$

    - for each cluster centre:
      - move the position of the centre to the mean of the points in that cluster (N_j is the number of points in cluster j):

$$\boldsymbol{\mu}_j = \frac{1}{N_j} \sum_{i=1}^{N_j} \mathbf{x}_i \tag{14.2}$$

  - until the cluster centres stop moving
- **Usage**
  - for each test point:
    - compute the distance to each cluster centre
    - assign the datapoint to the nearest cluster centre with distance

$$d_i = \min_j d(\mathbf{x}_i, \boldsymbol{\mu}_j). \tag{14.3}$$

K-means Clustering

Example:

The process is as follows:

Random Centroids: [(2,5) , (8,3) , (4, 5)]

Arbitrary point ***x*** : (0, 8)

Distance from point to each centroid: [3.61, 9.43, 5.00]

Point ***x*** is assigned to Centroid 1.

K-means Clustering

The **elbow method:**

K-means clustering **requires us to select K**, the number of clusters we want to group the data into.

The **elbow method** lets us **graph the inertia** (a distance-based metric) and visualize the point at which it starts decreasing linearly.

This point is referred to as the "elbow" and is a good estimate for the best value for K based on our data.

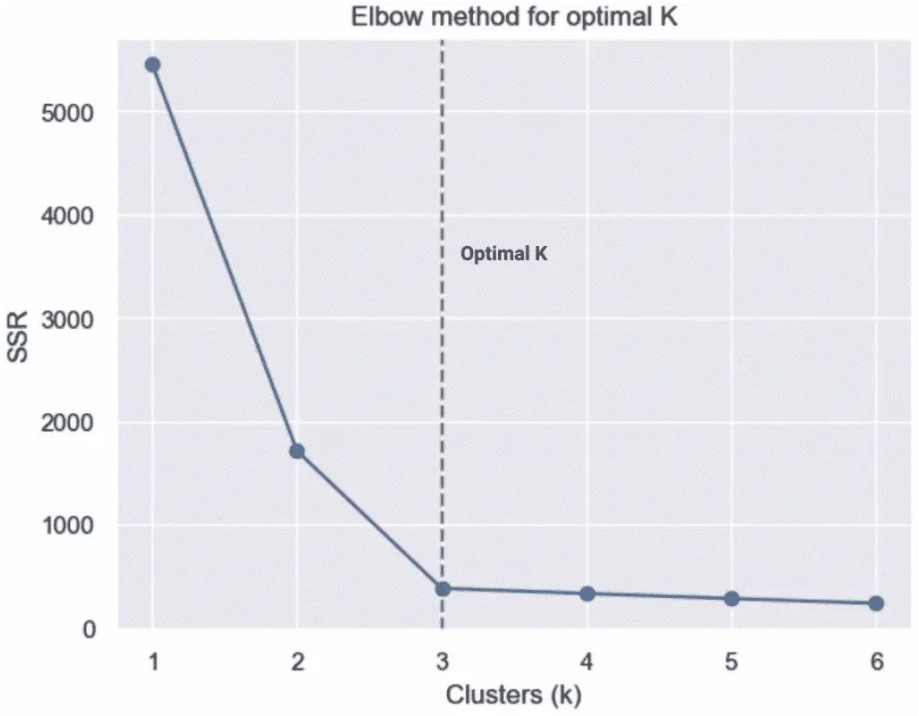
K-means Clustering

The **elbow method: Calculate inertia**

KMeans inertia, also known as **Sum of Squares Errors** (or SSE), calculates the sum of the distances of all points within a cluster from the centroid of the point.

(point_1 - mean)^2 + (point_2 - mean)^2 + (point_N - mean)^2

It is the **difference between the observed value and the predicted value**. It is calculated using the sum of the values minus the means, squared.



K-means Clustering

Dealing with Noise :

The mean average, which is central to the K-means algorithm, is **very susceptible to outliers**, i.e., **very noisy measurements**.

One way to avoid the problem is to **replace the mean average with the median,** meaning that it is not affected by outliers (the mean of (1, 2, 1, 2, 100) is 21.2, while the median is 2)

The only change that is needed to the algorithm is to **replace the computation of the mean with the computation of the median.** This is **computationally more expensive**, but it does remove noise effectively.

K-means Clustering

Normalizing the Data

When working with distance-based algorithms, like K-Means Clustering, we must normalize the data.

If we do not normalize the data, variables with different scaling will be weighted differently in the distance formula that is being optimized during training.

For example, if we were to include ***salary*** in the cluster, in addition to ***age*** and ***weight*****, *salary* would have an outsized impact on the optimizations** because its scale is **significantly larger** and wider than other variables.

K-means Clustering

K-Means is best suited for:

- Unsupervised learning problems
- With numeric data, does not work on categorical data
- Better suited for linear datasets

Advantages

- simple
- fast computation

Disadvantages

- need to define the clusters
- hard time handling outliers
- fail on non-linear datasets
- biased towards equal sized clusters

Next Week

- Neural Networks

Thank you for your participation ☺