Machine Learning : Statistical Approach

Independent Variables → Features
Dependent variable → The feature that we're trying to predict.

Regression vs Classification

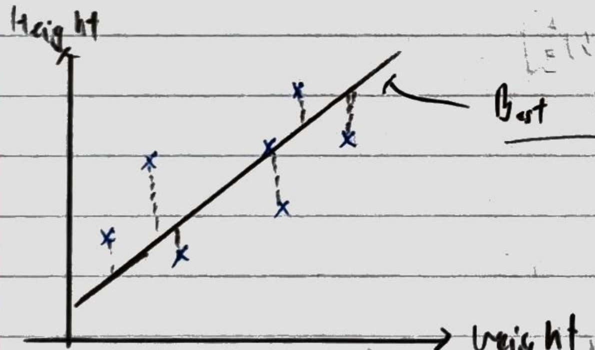
Linear Regression

Train [Linear Regression]

New weight → $\boxed{\text{Model}}$ → Height

Data set

Weight	Height (Dependent)
74	170 cm
80	180 cm
75	175.5 cm



Best Fit Line ~ Minimize the error [distance] between prediction and data points.

Equation
$h_\theta(x) = \theta_0 + \theta_1 x$

Cost Fn

$$J[\theta_0, \theta_1] = \frac{1}{2m} \sum_{i=1}^{m} [\underbrace{h_\theta(x)^i}_{\text{prediction}} - \underbrace{y^i}_{\text{True value}}]^2 \rightarrow (\text{Mean square error})$$

Convergence Algorithm

Repeat until convergence

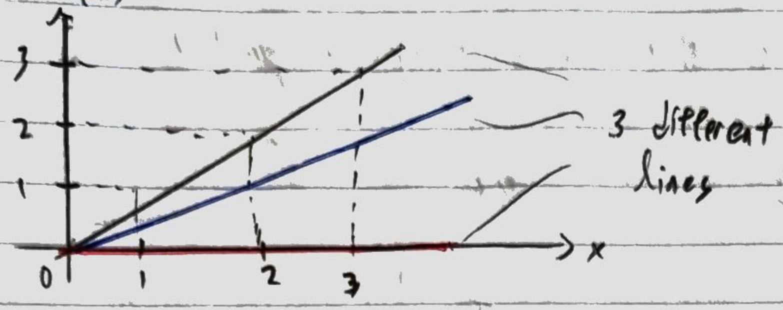
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J[\theta_0, \theta_1]$$

j = number of parameters

Graphs

Given Eqn $h_\theta(x) = \theta_1(x)$ Given data points $\{(1,1), (2,2), (3,3)\}$

Given Let θ_1 parameter $= 0, 0.5, 1$



- Consider the 'black' line with eqn $h_\theta(x) = 1x$, we get prediction $h_\theta(x)^i = \{1, 2, 3\}$

Calculate Cost Fn

$\theta_1 = 1$

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{3} \left[h_\theta(x)^i - y^i \right]^2$$

$$= \frac{1}{2(3)} \left[(1-1)^2 + (2-2)^2 + (3-3)^2 \right]$$

$$= 0$$

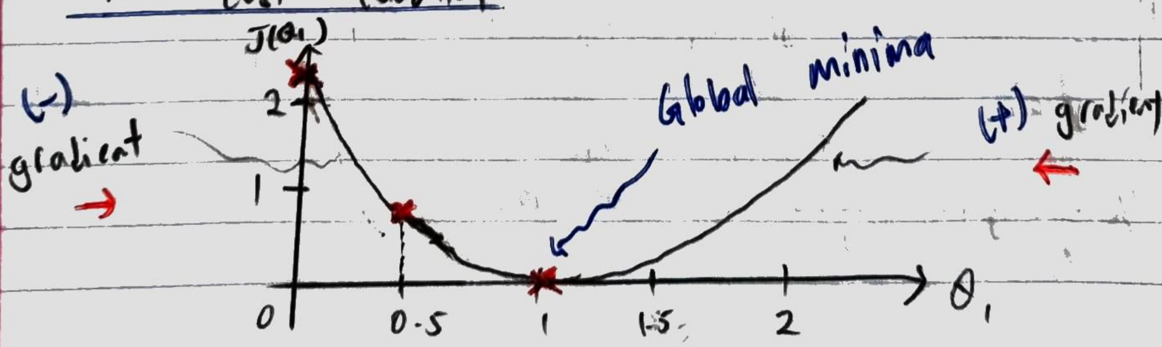
Blue line ~ $h_\theta(x) = 0.5x$ $\theta_1 = 0.5$

$$J(\theta_1) = \frac{1}{2(3)} \left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2 \right] \approx 0.58$$

Red line ~ $h_\theta(x) = 0[x] = 0$ $\theta_1 = 0$

$$J(\theta_1) = \frac{1}{2(3)} \left[(0-1)^2 + (0-2)^2 + (0-3)^2 \right] = 2.3$$

Plot cost function



Convergence Algorithm ~ Work iteratively, making ~~small adjust~~ adjustments to model parameters to improve performance
↳ minimize cost fn

[Gradient descent]

$$\theta_1 := \theta_1 - \underbrace{\alpha}_{\text{Learning rate}} \boxed{\frac{d}{d\theta_1} J(\theta_1)} \rightarrow \text{Gradient}$$

- New parameter for $h_\theta(x)$, use the new $h_\theta(x)$ to calculate cost function.
perform gradient descent to reach global minima

- with Negative $\left[\frac{d}{d\theta_i} J(\theta_i)\right]$, $\theta_1 \uparrow$ & $h_\theta(x) = \theta_1(x) \uparrow$
- with Positive $\left[\frac{d}{d\theta} J(\theta_i)\right]$, $\theta_1 \downarrow$ & $h_\theta(x) = \theta_1(x) \downarrow$

using these to calculate cost fn, and hopefully achieve global maxima / minima

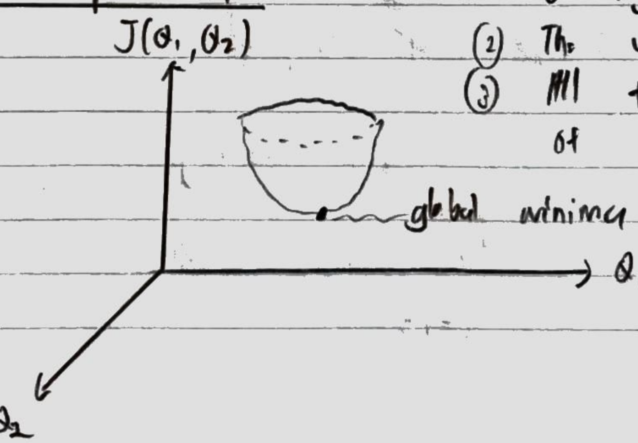
⇒ Learning rate determines the size of the steps taken during optimization, selecting appropriate value is crucial for achieving convergence and model accuracy.

⇒ Consider grid search with predefined range of learning rates to find the one that performs best.

L.R Assumptions

(1) The relationship between the feature set and target variable is linear
(2) The variance of the residual is constant
(3) All the observations are independent of one another

With 2 parameters



(4) The distribution of Y is assumed to be normal.

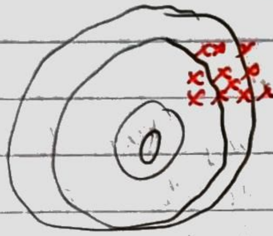
Definition of Bias and Variance

Training

Bias — How close the model's predicted values come to the true underlying f(x) values, with smaller being better

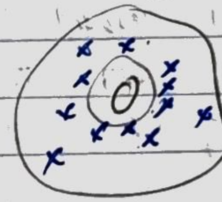
Variance — The extent to which model prediction error changed based on training inputs, with smaller being better. → Testing

Low variance & High bias
underfitting



→ Far-off from centre
but gathered data-points

High variance & Low bias
Overfitting

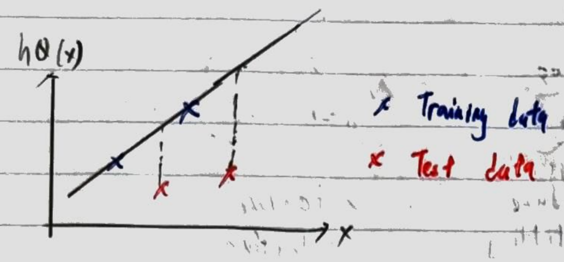


→ Around the center, but scattered.

Reducing Underfitting	Reducing Overfitting
① Increase model complexity ⇒ create more features	① Acquire more data ⇒ improve Generalization
② Reduce regularization parameter (λ)	② Feature selection ↓ model complexity
	③ Early stopping
	④ Ensemble Learning
	⑤ Increase regularization parameter (λ)

By Penalizing large coefficients

Ridge Regression [L_2 Regularization] → Reduce overfitting & underfitting



Problem
Cost fn $J(\theta_1, \theta_2) = 0$ for training data, but fails to generalize with the new data.

Overfitting

Low bias → model performs well with training data
High variance → Fails to perform well with test data

To make sure cost fn $\neq 0$. L2 Regularization do:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2 + \left[\lambda\left(\sum_{i=1}^{m}\theta_i^i\right)^2\right]$$

Adding a constant to cost fn

hyperparameter

slope

⇒ The impact of L2 is to penalize large coefficients, discouraging the model from relying too heavily on any particular feature.

L1 ~ Lasso Regression

Slope

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}[h_\theta(x^i) - y^i]^2 + \lambda\sum_{i=1}^{m}|\theta_i|$$ modulus

⇒ It tends to drive some of the coefficients (θ_i) to exactly zero.
Leads to sparse model where only a subset of features is deemed important effectively performing feature selection.

[not effective in reducing underfitting

⇒ Focus on ↓ overfitting.

$0 = -10^x$

$-\log_{10}(0.99)$

$-\log_{10}(0.01)$
$= \log_{10}(100)$

$0.01 = -10^x$
$-\frac{1}{100} = 10^x$
$x =$
$10^x = 100$
$x = 2$

Elastic net Regression [L_1 & L_2]

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left[h_\theta(x^i) - y^i\right]^2 + \underbrace{\lambda_1 \sum_{i=1}^{m}(\theta_i)^2}_{\text{Reduce overfitting}} + \underbrace{\lambda_2 \sum_{i=1}^{m}|\theta_i|}_{\text{Feature selection}}$$

Logistic Regression

* Designed for binary classification problem where the dependent variable is categorical with two possible outcomes [0, 1]

- Unlike linear regression, logistic regression is more robust in handling outliers due to its underlying logistic fn.

- LR is more flexible in capturing non-linear relationships, as it models the log-odds of the probability of the event occuring.

Equation

$$h_\theta(x) = \frac{1}{1+e^{-z}}$$

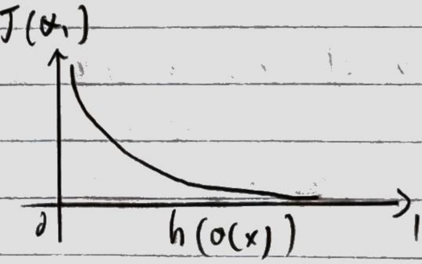
where $z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$

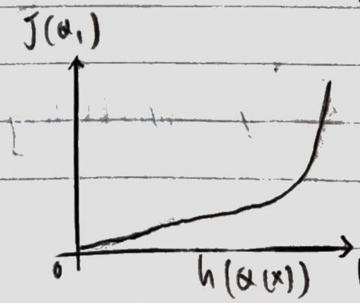
$0 \le h_\theta \le 1$

Cost Function

$\log\left[(h_\theta(x))^{-1}\right]$
$\|$

$$J(\theta_1) = \begin{cases} -\log(h_\theta(x)) & \text{if } y=1 \\ -\log\left[1-h_\theta(x)\right] & \text{if } y=0 \end{cases}$$



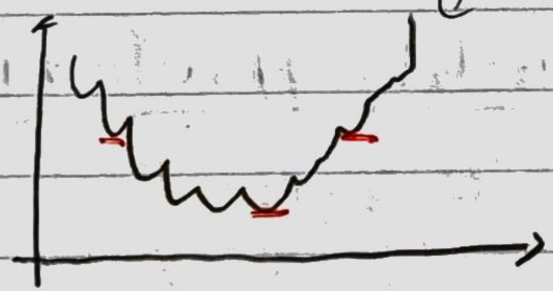


$$J[h_\theta(x^i), y^i] = -y^i \log[h_\theta(x^i)] - [1-y^i] \log[1 - h_\theta(x^i)]$$ where $y \in [0, 1]$

Gradient descent

$$\theta_j := \theta_j - \alpha \frac{d}{d\theta_j} \left[J[h_\theta(x^i), y^i] \right]$$

We cannot use $J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} [h_\theta(x)^i - y^i]^2$ for cost fn, because it will produce a non-convex function.

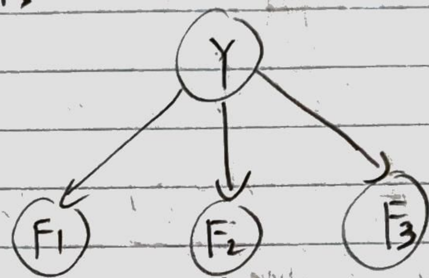


with local and global minima.

Naive Bayes

Assumption: Features are conditionally independent given class label (Y).

Given features: F_1, F_2, F_3



$$P(Y=y \mid F_1=f_1, F_2=f_2, F_3=f_3) = \frac{P(F_1=f_1, F_2=f_2, F_3=f_3 \mid Y=y)\, P(Y=y)}{P(F_1=f_1, F_2=f_2, F_3=f_3)}$$

$$= \frac{P(F_1=f_1 \mid Y=y)\, P(F_2=f_2 \mid Y=y)\, P(F_3=f_3 \mid Y=y)\, P(Y=y)}{P(F_1=f_1, F_2=f_2, F_3=f_3)}$$

$$P(F_1=f_1, F_2=f_2, F_3=f_3) = \sum_y P(F_1=f_1, F_2=f_2, F_3=f_3)\, P(Y=y)$$

$$= \sum_y P(F_1=f_1 \mid Y=y)\, P(F_2=f_2 \mid Y=y)\, P(F_3=f_3 \mid Y=y)\, P(Y=y)$$

Decision tree (without Pruning)

① Start at the root of the tree. Evaluate the information gain for each feature by calculating the entropy before and after the split for each possible value of the feature.

- Choose the feature that maximizes information gain as the root split.

② Split the dataset into subsets based on selected feature. Each subset corresponds to a unique value of the chosen feature

③ For each subset created by the split, repeat the process recursively

* Calculate the information gain for each feature in the subset
* Chose the feature that maximizes information gain and split the subset based on the chosen feature

④ Continue recursively until a stopping criterion is met. Stoppin criterion can be defined as:

⇒ Reaching a maximum depth
⇒ Having a minimum number of samples in a node
⇒ Achieving perfect purity [zero entropy] in a node for classification

⑤ When the stopping criterion is met for a node, assign a label to the leaf node. For classification tasks, this label is the majority class of the instances in the leaf.

⑥ To make prediction for new instance, traverse the tree from the root to a leaf node.

Entropy

$H(S)$ ~ Homogeneity of the target variable [class labels]

$$H(S) = -\sum_{i=1}^{c} p_i \log_2 [p_i]$$

c - number of class labels
p_i - proportion of instances in class i in the data

Information Gain

$$IG[S, A] = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

- $|S_v|$ is the number of instances in subset S_v after splitting on feature A
- $|S|$ is the total number of instances in dataset (S).
- $Values(A)$ represents the possible values of feature A.
- $H(S_v)$ is the entropy of subset S_v after splitting on feature A.

Gini Impurity ~ For faster computation

$$G(S) = 1 - \sum_{i=1}^{c} p_i^2$$

c — number of class labels
p_i - The proportion of instances in class i in dataset

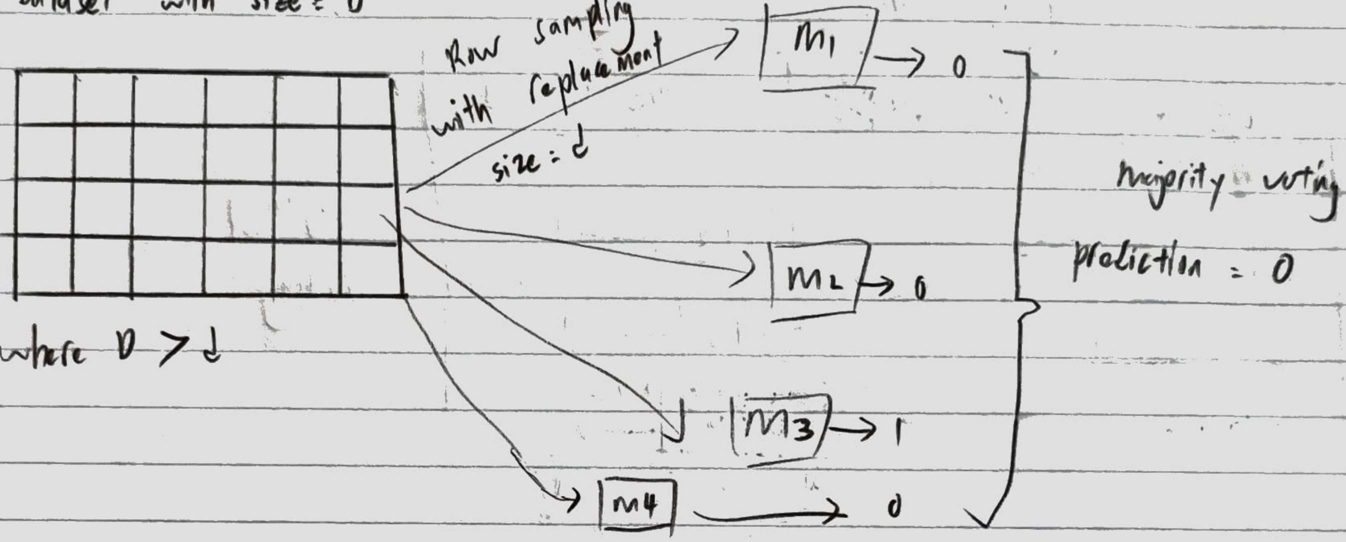
Gini Gian

$$Gini\ Gain(S, A) = G(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} G(S_v)$$

Bagging — Bootstrap Aggregating

* Sample with replacement to create bootstrap samples
  - Some observations may be selected multiple times
* Calculate statistical inference on each sample [median, mean...]
* Aggregates the result from each sample

Dataset with size = D



where D > d

① Create training data for each base learning algorithm by random sampling with replacement.
② A base learning algo is trained independently on each bootstrap sample
③ The final prediction is obtained by aggregating all the predictions
  - For regression, predictions are averaged
  - For classification, a majority vote is taken.

Advantage

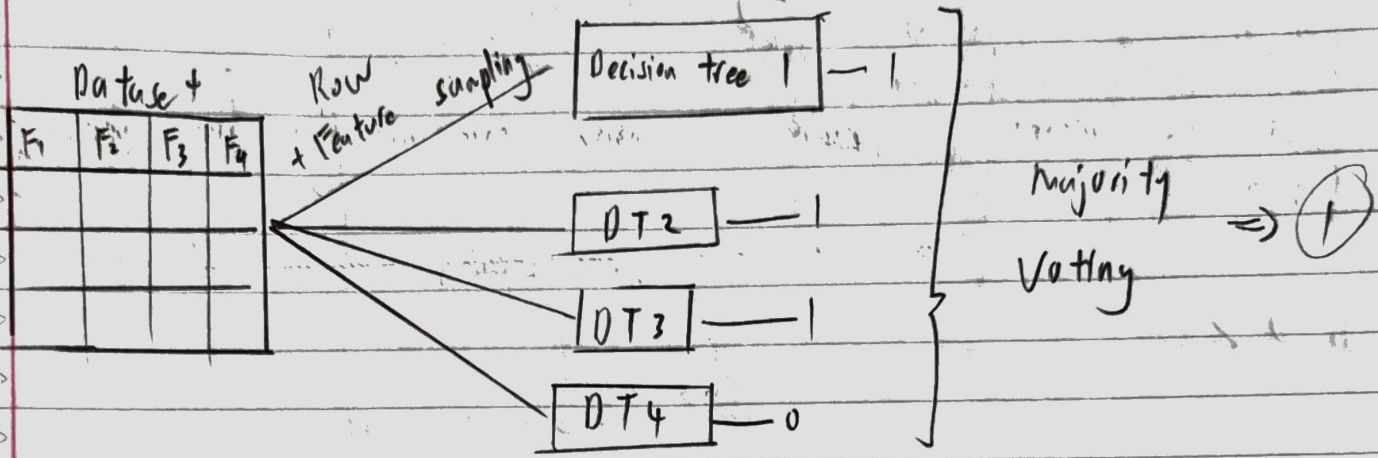
⇒ By training on different subsets of the data, bagging reduces the variance of the model. It helps to ↓ the likelihood of overfitting to the noise in the training data

Low bias

High variance ⇒ low V

Random Forest

Motivation : Random forest reduces overfitting compared to individual decision tree.



Advantages

- Improved Generalization ~ more robust and capable of generalizing to new data
- Provides a measure of feature importance, which helps identify the most influential features
- Training of individual trees in a Random Forest can be parallelized making it computationally efficient.
- Versatility ~ Random forest can handle both classification & Regression.

Boosting

$\rightarrow \boxed{M1} \rightarrow \boxed{M_2} \rightarrow \boxed{M_3} \rightarrow \boxed{M_4} \rightarrow$ output

- Boosting is an ensemble learning technique that combines the predictions of multiple weak learners to create a strong learner with improved accuracy.

- Builds models sequentially, with each subsequent model focusing on mistakes of the previous one.

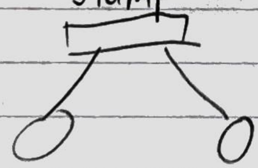
Adaboost

Adaptive boosting re-assigned weight to each instance, with higher weights assigned to incorrectly classified instances.

1. Assign equal weights to each data points.

2. Find the stump that does the best job classifying the samples by finding Gini index and selecting the one with lowest Gini Index.

Stump



3. Calculate the 'Total Error' and "Performance of the stump [α].

$$\text{Total Error} = \frac{\text{wrong output}}{\text{Total output}} \qquad \alpha = \frac{1}{2}\ln\left[\frac{1 - T.E}{T.E}\right]$$

(4) Update the previous sample weights:

$$\text{New weights} = \text{old weight} * [e^{\pm\alpha}]$$

⇒ Alpha will be negative when the sample is correctly classified
⇒ Alpha will be positive when the sample is miss-classified.

(5) Normalize the new sample weights and create Buckets [0 ~ weight]

(6) Create a new sample for the next stump. The Algo selects random number from 0-1. Since incorrectly classified records have higher sample weights, the probability of selecting mis-classified records are high.

(7) Reiterate the previous steps until a low training error is achieved.

K Nearest Neighbors (KNN) ~ Supervised Learning

① Read the entire training dataset, which consists input features and corresponding output labels

② Receive a new instance for which the prediction needs to be made.

③ Calculate the distance between the new instance and all the instances in the training set.

euclidean   manhattan

④ Choose K [hyperparameter] instances from the training dataset that are closest to the new instance.

⑤ Make the prediction for new instance with majority voting [Classification] or averaging [Reggression]

Key Considerations

Choice of K ~ A small K may leads to noise sensitivity
~ A large K may smooth out patterns in the data
~ Cross-validation is used to find an optimal value of K.

Normalization to ensure all features contribute to the distance calculation

KNN complexity increases with size of the training dataset.

Effective in scenario where the decision boundary is non-linear and complex

Unsupervised Learning

Machine learning technique where the algorithm is given input data without explicit output labels or target values.

K - Means Clustering

1. Choose K-initial cluster centroids $[\mu_1, \mu_2, \ldots, \mu_k]$ where μ_i represents the centroid of the i-th cluster.

2. For each data point x^i, find the nearest centroid c^i using the Euclidean distance

$$c^{(i)} = \arg\min_j \| x^i - \mu_j \|^2$$

* Assign the data point to the cluster associated with the nearest centroid.

3. Update each centroid μ_j as the mean of all data points assigned to cluster j.

$$\mu_j = \frac{1}{|C_j|} \sum_{i \in C_j} x^i$$

$|C_j|$ ~ Number of data points assigned to cluster j.

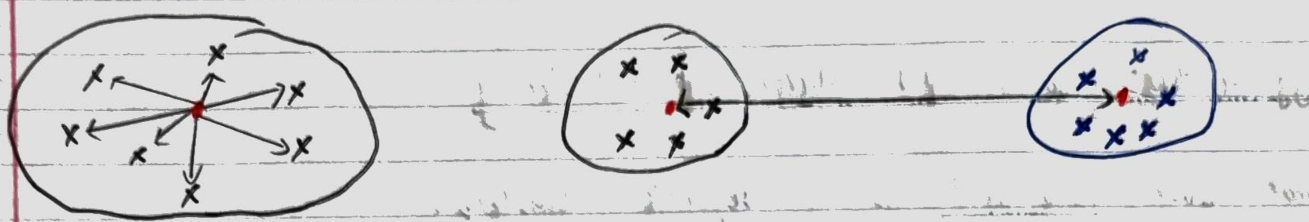
→ Summing up the coordinates of all the data points in the cluster, and dividing by the number of clusters.

4. Repeat steps 2 and 3 until convergence when the centroids no longer change significantly.

Properties

① All the data points in a cluster should be similar to each other.

② The data points from different clusters should be as different as possible.



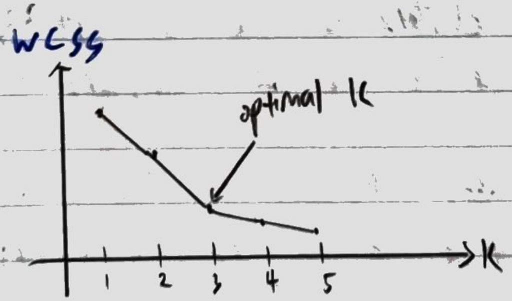
Intracluster distance - sum of distances of all the points within a cluster from the centroid

Intercluster distance - The distance between the centroids of 2 different clusters.

③ Maximize Dunn index = $\dfrac{\min[\text{intercluster } d] \uparrow\uparrow}{\max[\text{intracluster } d] \downarrow\downarrow}$

Within-Cluster sum of squares [WCSS] $= \sum_{j=1}^{K} \sum_{i \in C_j} ||x^i - \mu_j||^2$

Determining the Number of Clusters (K) : **Elbow Method**



>3 might overfit and increase computational power.

~~Hier~~ Hierarchical Clustering

→ Techniques that builds a hierarchy of clusters by successively merging or splitting existing clusters.

Agglomerative [bottom-up] Hierarchical Clustering

1. Treat each data point as a separate clusters.

2. Calculate the pairwise distances [Euclidean] between all clusters or data points.

3. Identify the two clusters or data points that are closest to each other based on the computed distances.

4. Merge these two clusters into a new cluster.

5. Recalculate the distances between the new cluster and all the other clusters

- Depends on the linkage method.

Single Linkage - The distance between 2 clusters is the shortest distance between any two points in the 2 clusters.

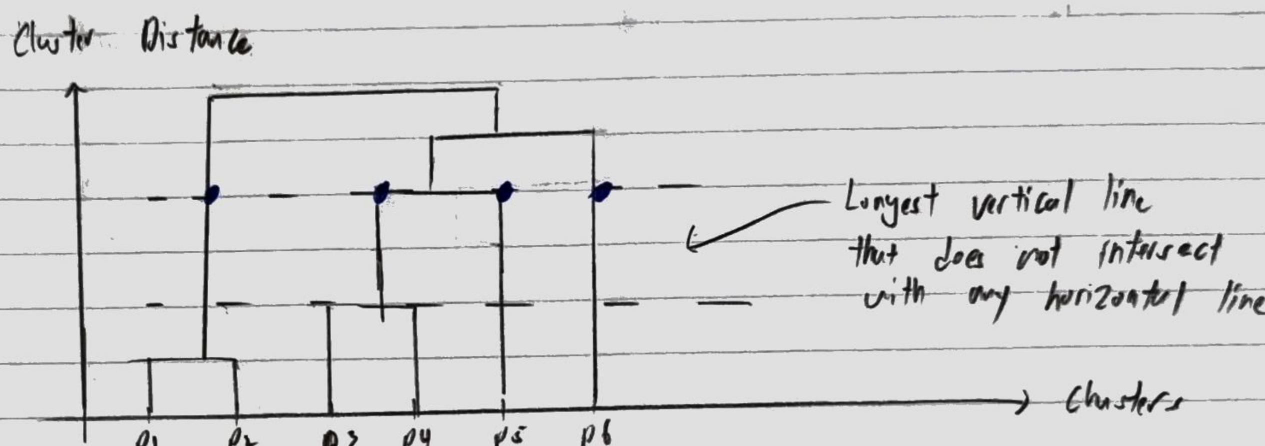
Complete Linkage - The distance between 2 clusters is the longest distance between any two points in the 2 clusters.

Average Linkage - Distance between 2 clusters is the average distance between all pairs of points in 2 clusters

$$d[A, B] = \frac{1}{|A| \cdot |B|} \sum_{i \in A} \sum_{j \in B} \text{distance}[i, j]$$

⑥ Repeat steps ④ and ⑤ until only a single cluster [the root of the dendogram] remains.

Dendogram



4 clusters is the estimate of optimal number of clusters.

* The longest vertical line implies the maximum Intercluster distance.

Silhouette Score

A measure of how well-separated the clusters are. Range from −1 to 1.

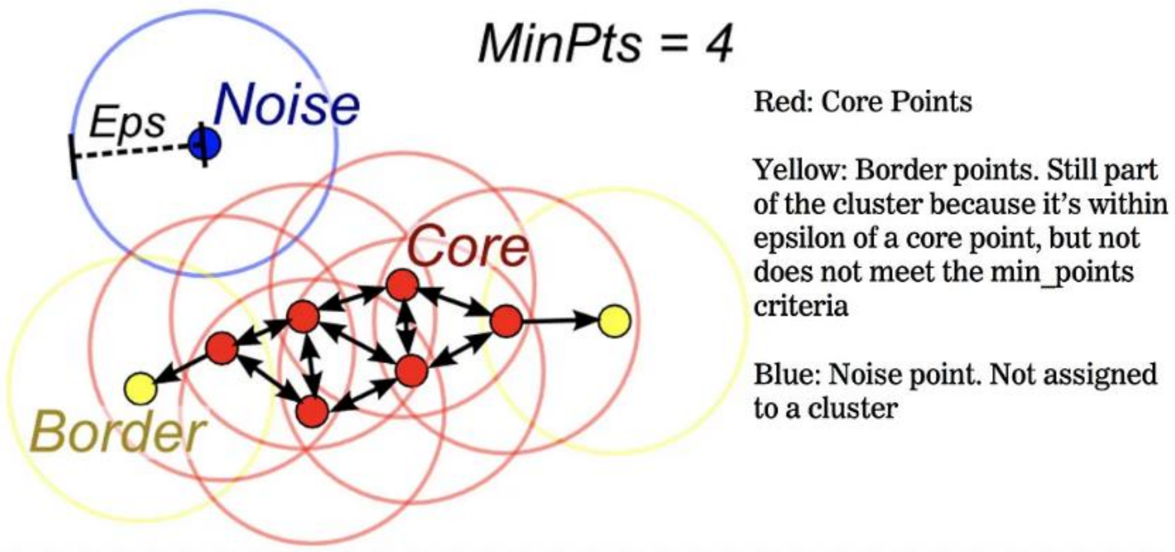
$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

$a(i)$ ~ Average distance from point i to other points in the same Cluster:

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i} d(i,j) \quad \text{where } j \neq i$$

$b(i)$ – Average distance from point i to points in the nearest neighboring cluster.

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j)$$



Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a clustering algorithm used for identifying clusters in a dataset based on the density of data points. Unlike traditional clustering algorithms like K-Means, DBSCAN does not require a predetermined number of clusters and can discover clusters of arbitrary shapes. DBSCAN is particularly effective in handling noise and outliers in the data.

Key Concepts in DBSCAN:

1. **Core Points**:

- A data point is considered a core point if it has at least a specified number of neighboring points (minPts) within a defined radius (ε).

2. **Border Points**:

- A data point is considered a border point if it has fewer neighboring points than minPts but lies within the radius (ε) of a core point.

3. **Noise Points**:

- A data point that is neither a core nor a border point is considered a noise point or an outlier.

4. **Directly Density-Reachable**:

- Two points A and B are directly density-reachable if B is a core point and A is within the radius of B.

5. **Density-Reachable**:

- Two points A and B are density-reachable if there is a sequence of points P_1, P_2, …., P_n such that P_1 = A, P_n = B, and each P_i is directly density-reachable from P_{i-1}.

DBSCAN Algorithm Steps:

1. **Initialization**:

- Choose an arbitrary data point from the dataset.

2. **Core Point Identification**:

- If the chosen point is a core point (has at least minPts neighbors within (ε), a new cluster is formed.

3. **Density-Reachable Expansion**:

- Expand the cluster by adding all directly density-reachable points to it.

4. **Repeat Steps 2-3**:

- Repeat the process until no more core points can be found.

5. **Noise Point Handling**:

- Any remaining data points that have not been assigned to a cluster are treated as noise points.

Advantages of DBSCAN:

- Ability to Identify Arbitrary-Shaped Clusters: DBSCAN can find clusters with irregular shapes and is not sensitive to the number of clusters in advance.

- Robust to Outliers: The algorithm is robust to noise and outliers, classifying them as noise points rather than forcing them into a cluster.

- No Need for Predefined Number of Clusters: DBSCAN does not require specifying the number of clusters beforehand, making it more flexible.

Parameters in DBSCAN:

ε - The radius within which to search for neighboring points.

minPts (Minimum Points): The minimum number of data points required to form a dense region.

Metrics for Measuring DBSCAN's Performance:

Silhouette Score: The silhouette score is calculated utilizing the mean intra- cluster distance between points, AND the mean nearest-cluster distance. For instance, a cluster with a lot of data points very close to each other (high density) AND is far away from the next nearest cluster (suggesting the cluster is very unique in comparison to the next closest), will have a strong silhouette score. A silhouette score ranges from -1 to 1, with -1 being the worst score possible and 1 being the best score. Silhouette scores of 0 suggest overlapping clusters.