UNIT 3

Combining Multiple Learners– Model Combination schemes, voting, Bagging, Boosting.

Unsupervised Learning: K-Means, Expectation Maximization Algorithm, supervised learning after clustering, spectral clustering, and Choosing number of clusters.

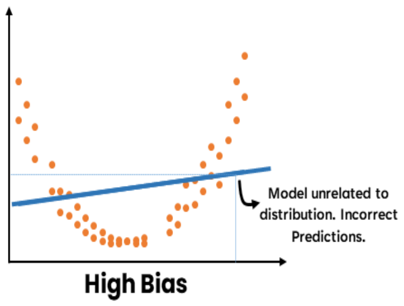
How Ensemble Learning Works?

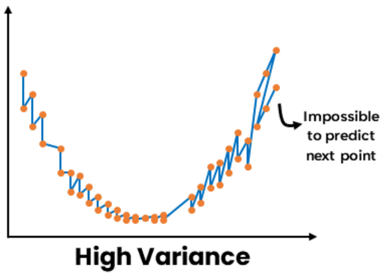
Ensemble learning is a learning method that consists of combining multiple machine learning models. A problem in machine learning is that individual models tend to perform poorly. In other words, they tend to have low prediction accuracy. To mitigate this problem, we combine multiple models to get one with a better performance.

The individual models that we combine are known as **weak learners**. We call them weak learners because they have a high bias or high variance.

Because they either have high bias or variance, weak learners cannot learn efficiently and perform poorly.

A high-bias model results from not learning data well enough. It is not related to the distribution of the data. Hence future predictions will be unrelated to the data and thus incorrect.



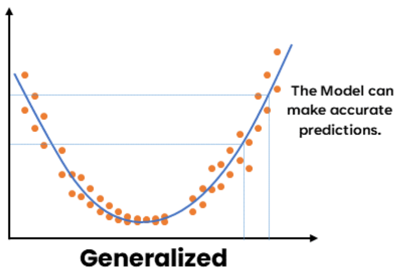


A high variance model results from learning the data too well. It varies with each data point. Hence it is impossible to predict the next point accurately.

Both high bias and high variance models thus cannot generalize properly. Thus, weak learners will either make incorrect generalizations or fail to generalize altogether. Because of this, the predictions of weak learners cannot be relied on by themselves.

As we know from the bias-variance trade-off, an under fit model has high bias and low variance, whereas an over fit model has high variance and low bias. In either case, there is no balance between bias and variance. For there to be a balance, both the bias and variance need to be low. Ensemble learning tries to balance this bias-variance trade-off by reducing either the bias or the variance.

Ensemble learning will aim to reduce the bias if we have a weak model with high bias and low variance. Ensemble learning will aim to reduce the variance if we have a weak model with high variance and low bias. This way, the resulting model will be much more balanced, with low bias and variance. Thus, the resulting model will be known as a strong learner. This model will be more generalized than the weak learners. It will thus be able to make accurate predictions.



Ensemble learning improves a model's performance in mainly three ways:

- By reducing the variance of weak learners
- By reducing the bias of weak learners,
- By improving the overall accuracy of strong learners.

Bagging is used to reduce the variance of weak learners. Boosting is used to reduce the bias of weak learners. Stacking is used to improve the overall accuracy of strong learners.

Bagging, boosting and stacking are the three most popular ensemble learning techniques. Each of these techniques offers a unique approach to improving predictive accuracy. Each technique is used for a different purpose, with the use of each depending on varying factors. Although each technique is different, many of us find it hard to distinguish between them. Knowing when or why we should use each technique is difficult.

Ensemble learning can be performed in two ways:

1. **Sequential ensemble,** popularly known as ***boosting***, here the weak learners is sequentially produced during the training phase. The performance of the model is improved by assigning a higher weightage to the previous, incorrectly classified samples. An example of boosting is the AdaBoost algorithm.
2. **Parallel ensemble**, popularly known as ***bagging***, here the weak learners are produced parallelly during the training phase. The performance of the model can be increased by parallelly training a number of weak learners on bootstrapped data sets. An example of bagging is the Random Forest algorithm.

Types of Ensemble Methods

- Voting
- Bagging (Decrease Variance)
- Boosting (Decrease Bias)
- Stacking (Improve Predictions)

Combining Multiple Learners: -

Though the different learning algorithms are generally successful, no one single algorithm is always the most accurate. Now, we are going to discuss models composed of multiple learners that complement each other so that by combining them, we attain higher accuracy.

Model Combination schemes: -

There are also different ways the multiple base-learners are combined to generate the final output

Multiexpert combination: -

Multiexpert combination methods have base-learners that work in parallel. These methods can in turn be divided into two:

- In the global approach, also called learner fusion, given an input, all base-learners generate an output and all these outputs are used. Examples are **voting** and **stacking**.
- In the local approach, or learner selection, for example, in mixture of experts, there is a **gating model**, which looks at the input and chooses one (or very few) of the learners as responsible for generating the output.

Multistage combination: -

Multistage combination methods use a serial approach where the next base-learner is trained with or tested on only the instances where the previous base-learners are not accurate enough. The idea is that the base-learners (or the different representations they use) are sorted in increasing complexity so that a complex base-learner is not used (or its complex representation is not extracted) unless the preceding simpler base-learners are not confident. An example is **cascading**.

Let us say that we have L base-learners. We denote by $d_j(x)$ the prediction of base-learner M_j given the arbitrary dimensional input x. In the case of multiple representations, each M_j uses a different input representation x_j. The final prediction is calculated from the predictions of the base-learners:

$$y = f(d_1, d_2, \ldots, d_L | \Phi)$$

where f (·) is the combining function with Φ denoting its parameters.

When there are K outputs, for each learner there are $d_{ji}(x)$, i = 1,...,K, j = 1,...,L, and, combining them, we also generate K values, y_i, i= 1,...,K and then for example in classification, we choose the class with the maximum y_i value:

$$\text{Choose } C_i \text{ if } y_i = \max_{k=1}^{K} y_k$$

Voting: -

The simplest way to combine multiple classifiers is by voting, which corresponds to taking a linear combination of the learners (see figure 17.1):

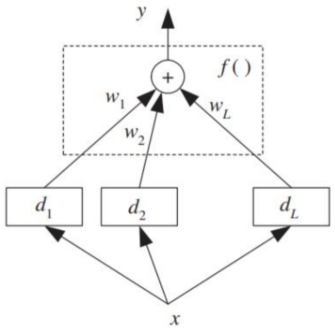


Figure 17.1 Base-learners are d_j and their outputs are combined using $f(\cdot)$. This is for a single output; in the case of classification, each base-learner has K outputs that are separately used to calculate y_i, and then we choose the maximum. Note that here all learners observe the same input; it may be the case that different learners observe different representations of the same input object or event.

This is also known as ensembles and linear opinion pools. In the simplest case, all learners are given equal weight and we have simple voting that corresponds to taking an average. Still, taking a (weighted) sum is only one of the possibilities and there are also other combination rules, as shown in table 17.1. If the outputs are not posterior probabilities, these rules require that outputs be normalized to the same scale.

Table 17.1 Classifier combination rules

Rule	Fusion function $f(\cdot)$
Sum	$y_i = \frac{1}{L}\sum_{j=1}^{L} d_{ji}$
Weighted sum	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$

An example of the use of these rules is shown in table 17.2, which demonstrates the effects of different rules. Sum rule is the most intuitive and is the most widely used in practice. Median rule is more robust to outliers; minimum and maximum rules are pessimistic and optimistic, respectively. With the product rule, each learner has veto power; regardless of the other ones, if one learner has an output of 0, the overall output goes to 0. Note that after the combination rules, y_i do not necessarily sum up to 1.

Table 17.2 Example of combination rules on three learners and three classes

	C_1	C_2	C_3
d_1	0.2	0.5	0.3
d_2	0.0	0.6	0.4
d_3	0.4	0.4	0.2
Sum	0.2	**0.5**	0.3
Median	0.2	**0.5**	0.4
Minimum	0.0	**0.4**	0.2
Maximum	0.4	**0.6**	0.4
Product	0.0	**0.12**	0.032

The voting classifier is an ensemble learning method that combines several base models to produce the final optimum solution. The base model can independently use different algorithms such as KNN, Random forests, Regression, etc., to predict individual outputs. This brings diversity in the output, thus called **Heterogeneous ensembling.** In contrast, if base models use the same algorithm to predict separate outcomes, this is called **Homogeneous ensembling.**

Voting Classifier is an estimator that combines models representing different classification algorithms associated with individual weights for confidence.

The Voting classifier estimator built by combining different classification models turns out to be stronger meta-classifier that balances out the individual classifiers' weaknesses on a particular dataset.

Voting classifier takes **majority voting based on weights** applied to the **class** or **class probabilities** and assigns a class label to a record based on majority vote.
The ensemble classifier prediction can be mathematically represented as the following:

$$\hat{y} = \arg\max_i \sum_{j=1}^{m} w_j \chi_A(C_j(x) = i)$$

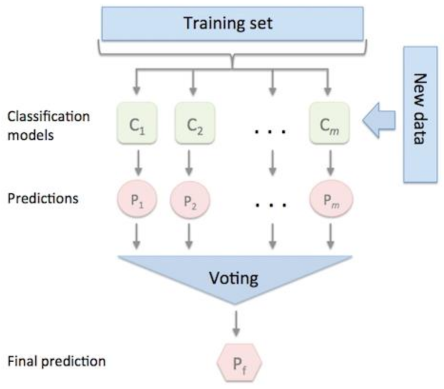
Weighted Majority Vote for Ensemble Classifier
In above equation, C_j represents the classifer, w_j represents the weight associated with the prediction of the classifier. There are two different types of voting classifier.

1. Hard voting classifier and
2. Soft voting classifier

Hard Voting Classifiers – How do they work?

Hard voting is also known as majority voting. The base model's classifiers are fed with the training data individually. The models predict the output class independent of each other. The output class is a class expected by the majority of the models.

Hard voting classifier classifies input data based on the **mode** of all the predictions made by different classifiers. The majority voting is considered differently when weights associated with the different classifiers are equal or otherwise.



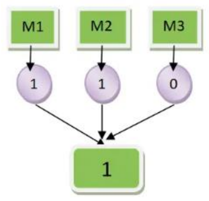
In the above figure, P_f is the class predicted by the majority of the classifiers C_m.
Majority Voting based on equal weights: When majority voting is taken based equal weights, **mode** of the predicted label is taken. Let's say there are 3 classifiers, clf1, clf2, clf3. For a particular data, the prediction is [1, 1, 0]. In case, the weights assigned to the classifiers are equal, the mode of the prediction is taken. Thus, the mode of [1, 1, 0] is 1 and hence the predicted class to the particular record becomes class 1. For equal weights, the equation in fig 1 gets simplified to the following:

$$\hat{y} = \arg\max_i \sum_{j=1}^{m} w_j \chi_A(C_j(x) = i)$$

$$\text{mode } clf_1, clf_2, clf_3$$

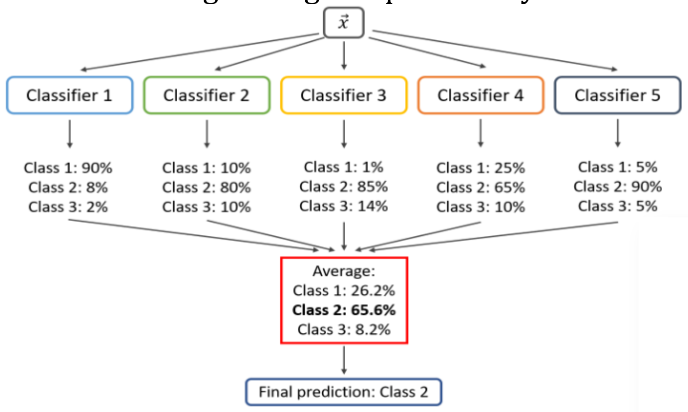
$$\text{mode}[1, 1, 0] = 1$$

Diagrammatically, this is how the hard voting classifier with equal weights will look like:



Soft voting: -

In soft voting, Classifiers or base models are fed with training data to predict the classes out of m possible courses. Each base model classifier independently assigns the probability of occurrence of each type. In the end, the average of the possibilities of each class is calculated, and the final output is the class having the highest probability.


$\vec{x}$
Classifier 1
Classifier 2
Classifier 3
Classifier 4
Classifier 5
Class 1: 90% Class 2: 8% Class 3: 2%
Class 1: 10% Class 2: 80% Class 3: 10%
Class 1: 1% Class 2: 85% Class 3: 14%
Class 1: 25% Class 2: 65% Class 3: 10%
Class 1: 5% Class 2: 90% Class 3: 5%
Average: Class 1: 26.2% Class 2: 65.6% Class 3: 8.2%
Final prediction: Class 2

Soft voting classifier classifies input data based on the **probabilities** of all the predictions made by different classifiers. Weights applied to each classifier get applied appropriately based on the equation given.

$$\hat{y} = \arg\max_i \sum_{j=1}^{m} w_j \chi_A(C_j(x) = i)$$

Let's understand this using an example. Let's say there are two binary classifiers clf1, clf2 and clf3. For a particular record, the classifiers make the following predictions in terms of probabilities in favour of classes [0,1]:

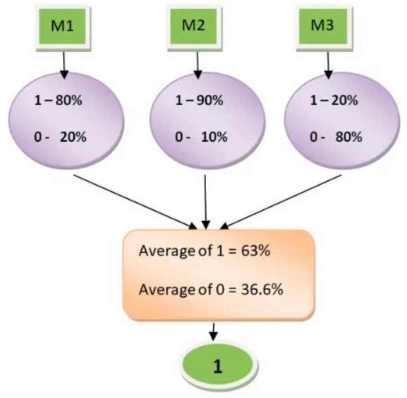
clf1 -> [0.2, 0.8], clf2 -> [0.1, 0.9], clf3 -> [0.8, 0.2]

With equal weights, the probabilities will get calculated as the following:

Prob of Class 0 = 0.33*0.2 + 0.33*0.1 + 0.33*0.8 = 0.363

Prob of Class 1 = 0.33*0.8 + 0.33*0.9 + 0.33*0.2 = 0.627

The probability predicted by ensemble classifier will be [36.3%, 62.7%]. The class will most likely by class 1 if the threshold is 0.5. This is how a soft voting classifier with equal weights will look like:



Soft Voting Classifier with Equal Weights

With unequal weights [0.6, 0.4], the probabilities will get calculated as the following:

Prob of Class 0 = 0.6*0.2 + 0.4*0.6 = 0.36

Prob of Class 1 = 0.6*0.8 + 0.4*0.4 = 0.64

The probability predicted by ensemble classifier will be [0.36, 0.64]. The class wills most likely by class 1 if the threshold is 0.5.

Conclusion

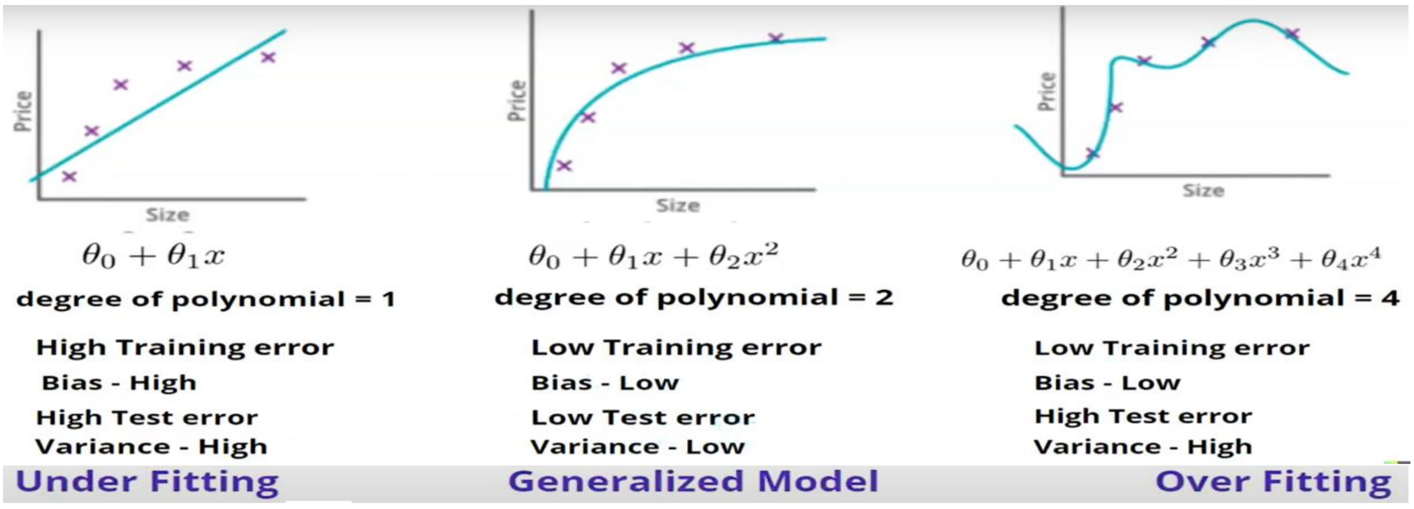
- Voting classifier can be seen as a stronger **meta-classifier** that balances out the individual classifiers' weaknesses on a particular dataset.
- Voting classifier is an **ensemble classifier** which takes input as two or more estimators and classifies the data based on majority voting.
- **Hard voting classifier** classifies data based on **class labels** and the **weights** associated with each classifier
- **Soft voting classifier** classifies data based on the **probabilities** and the **weights** associated with each classifier.
- **Hard-voting ensembles output the mode** of the base classifiers' predictions, whereas **soft-voting ensembles average predicted probabilities** (or scores).

Bais, Variance, Under Fitting & Over Fitting: -

A supervised machine learning model aims to train itself on the input variables (X) in such a way that the predicted values(Y) are as close to the actual values as possible. This difference between the actual values and predicted values is the error and it is used to evaluate the model. The error for any supervised machine learning algorithm comprises of 3 parts:

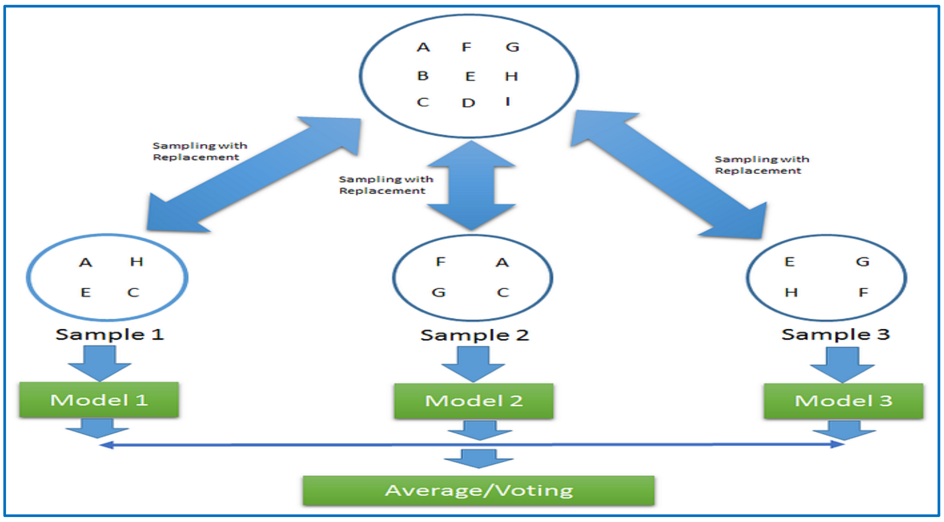
1. Bias error
2. Variance error
3. The noise

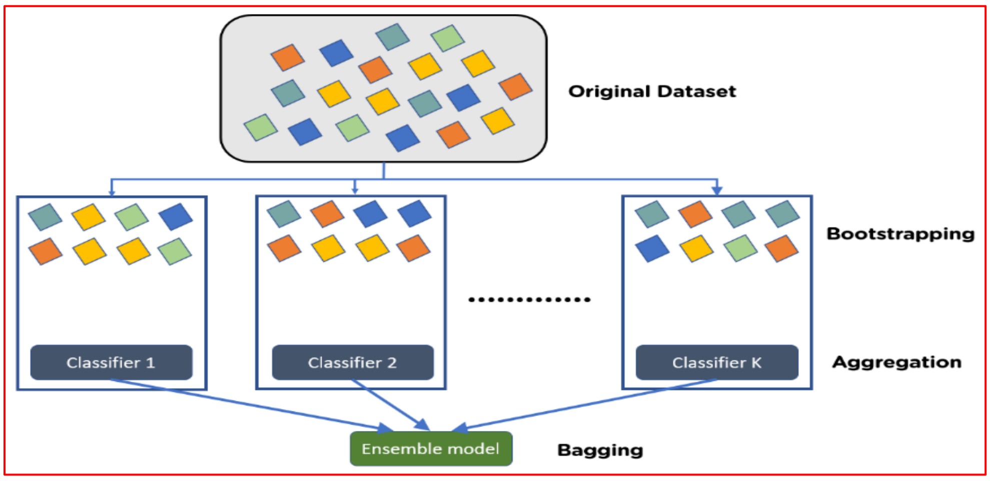
1. **Bias error: -** Our model will not be trained well with the training data. There will be high training error when we train our model with the data.
2. **Variance error: -** If you train your data on training data and obtain a very low error, upon changing the data and then training the same previous model you experience high error, this is variance.
3. **The noise: -** While we are collecting the data, if we have outliers then the data is noisy data.



Bagging: -

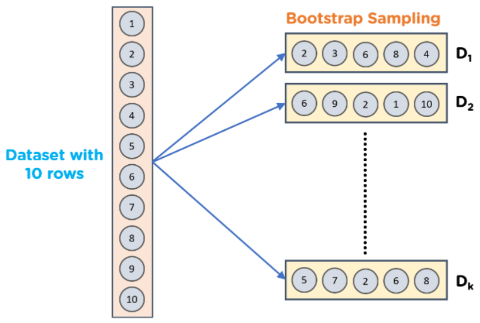
Bagging, also known as Bootstrap aggregating, is an ensemble learning technique that helps to improve the performance and accuracy of machine learning algorithms. It is used to deal with bias-variance trade-offs and reduces the variance of a prediction model. Bagging avoids over fitting of data and is used for both regression and classification models, specifically for decision tree algorithms.





What Is Bootstrapping?

Bootstrapping is the method of randomly creating samples of data out of a population with replacement to estimate a population parameter.



Aggregation

In order to take into account all the future effects, model predictions undergo aggregation to integrate them for the final forecast. Based on the total number of effects or on the likelihood of projections obtained from the bootstrapping of and model in the process, the aggregation can be performed.

Stages of Bagging

- **Bootstrapping:** This is a mathematical method used to produce random samples or bootstrap samples with replacement.
- **Model fitting:** We create models on bootstrap samples at this point. Usually, to construct the models, the same algorithm is use. Nevertheless, there is no limitation on using multiple algorithms.

- **Combining models:** This step entails all models being mixed and an average is taken. For eg, if a decision tree classifier has been implemented, then the probability that comes out of each classifier is averaged.

Steps to Perform Bagging: -

- Consider there are n observations and m features in the training set. You need to select a random sample from the training dataset without replacement
- A subset of m features is chosen randomly to create a model using sample observations
- The feature offering the best split out of the lot is used to split the nodes
- The tree is grown, so you have the best root nodes
- The above steps are repeated n times. It aggregates the output of individual decision trees to give the best prediction

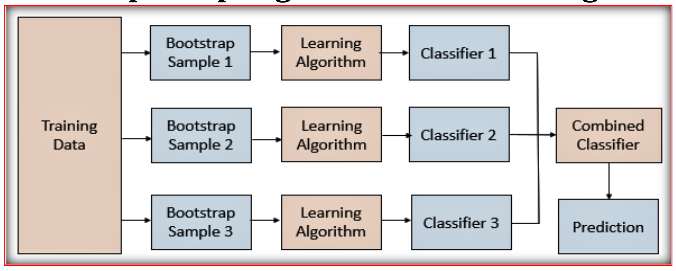
The predictions are aggregated using either max voting or averaging.

- **Max Voting** is commonly used for classification problems. It consists of taking the mode of the predictions (the most occurring prediction). It is called voting because like in election voting, the premise is that 'the majority rules'. Each model makes a prediction. A prediction from each model counts as a single 'vote'. The most occurring 'vote' is chosen as the representative for the combined model.
- **Averaging** is generally used for regression problems. It involves taking the average of the predictions. The resulting average is used as the overall prediction for the combined model.

Advantages of Bagging in Machine Learning

- Bagging minimizes the overfitting of data
- It improves the model's accuracy
- It deals with higher dimensional data efficiently

Bootstrap Sampling in Machine Learning



Boosting in Machine Learning

Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models is added. For example, if a cat-identifying model has been trained only on images of white cats, it may occasionally misidentify a black cat. Boosting tries to overcome this issue by training multiple models sequentially to improve the accuracy of the overall system.

Why is boosting important?

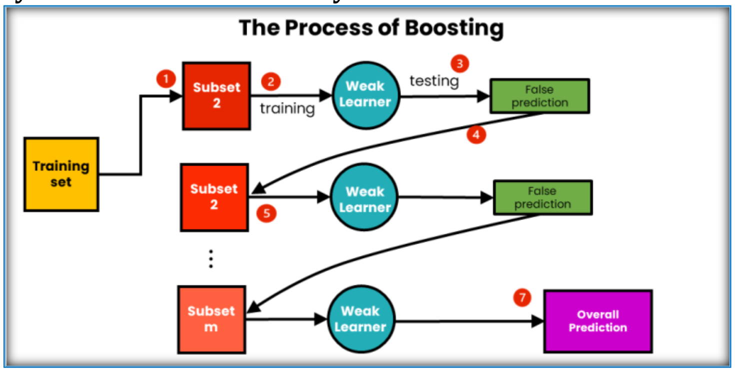
Boosting improves machine models' predictive accuracy and performance by converting multiple weak learners into a single strong learning model. Machine learning models can be weak learners or strong learners:

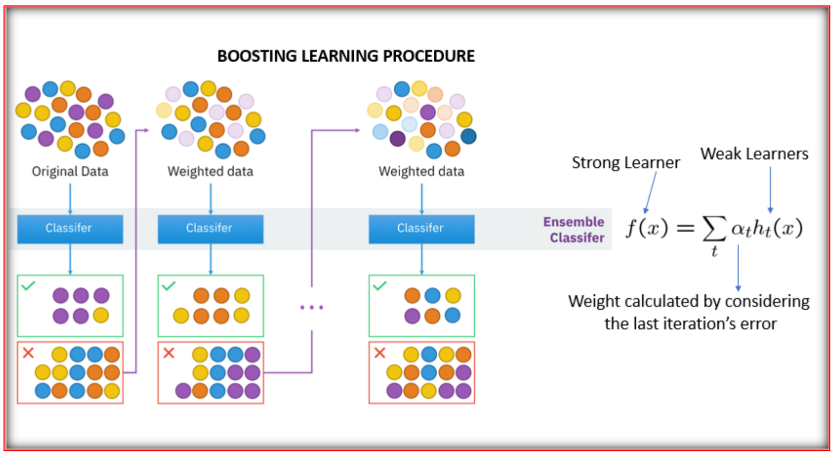
Weak learners

Weak learners have low prediction accuracy, similar to random guessing. They are prone to overfitting—that is, they can't classify data that varies too much from their original dataset. For example, if you train the model to identify cats as animals with pointed ears, it might fail to recognize a cat whose ears are curled.

Strong learners

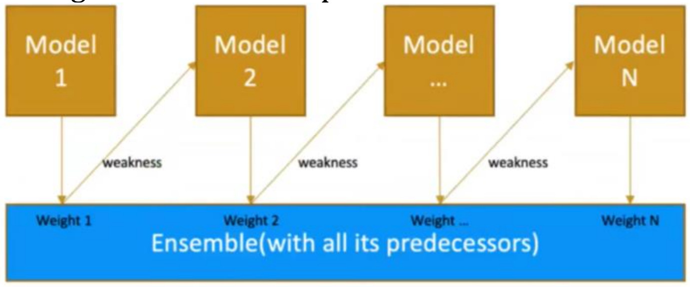
Strong learners have higher prediction accuracy. Boosting converts a system of weak learners into a single strong learning system. For example, to identify the cat image, it combines a weak learner that guesses for pointy ears and another learner that guesses for cat-shaped eyes. After analyzing the animal image for pointy ears, the system analyzes it once again for cat-shaped eyes. This improves the system's overall accuracy.




BOOSTING LEARNING PROCEDURE
Original Data
Weighted data
Weighted data
Classifer
Classifer
Classifer
Ensemble Classifer
Strong Learner
Weak Learners
$f(x) = \sum_t \alpha_t h_t(x)$
Weight calculated by considering the last iteration's error

How Boosting Algorithm Works?

The basic principle behind the working of the boosting algorithm is to generate multiple weak learners and combine their predictions to form one strong rule. These weak rules are generated by applying base Machine Learning algorithms on different distributions of the data set. These algorithms generate weak rules for each iteration. After multiple iterations, the weak learners are combined to form a strong learner that will predict a more accurate outcome.



Types of Boosting: -

Boosting methods are focused on iteratively combining weak learners to build a strong learner that can predict more accurate outcomes. As a reminder, a weak learner classifies data slightly better than random guessing. This approach can provide robust prediction problem results, outperform neural networks, and support vector machines for tasks.

Boosting algorithms can differ in how they create and aggregate weak learners during the sequential process. Three popular types of boosting methods include:

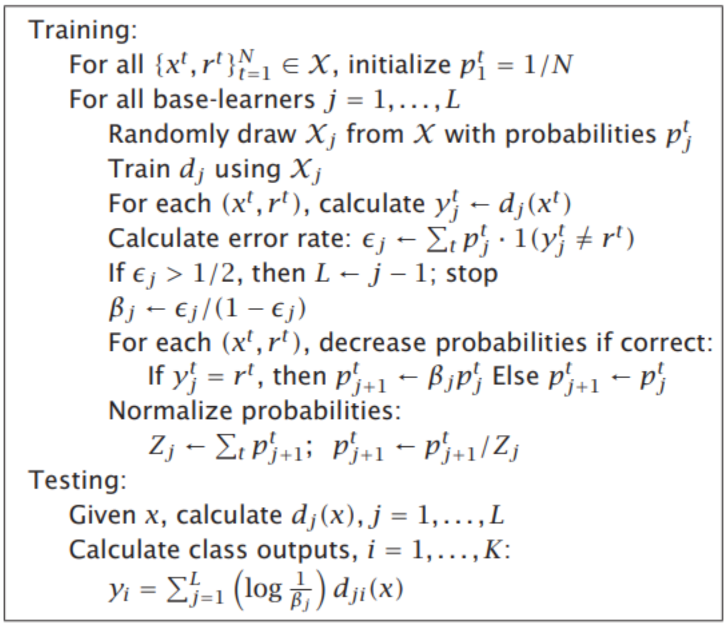
1. **Adaptive boosting or AdaBoost:**
2. **Gradient Boosting**
3. **Extreme gradient boosting or XGBoost**

AdaBoost: -

AdaBoost was the first really successful boosting algorithm developed for the purpose of binary classification. *AdaBoost* is short for *Adaptive Boosting* and is a very popular boosting technique that combines multiple "weak classifiers" into a single "strong classifier".

AdaBoost, also called Adaptive Boosting, is a technique in Machine Learning used as an Ensemble Method. The most common estimator used with AdaBoost is decision trees with one level which means Decision trees with only 1 split. These trees are also called Decision Stumps.

What this algorithm does is that it builds a model and gives equal weights to all the data points. It then assigns higher weights to points that are wrongly classified. Now all the points with higher weights are given more importance in the next model. It will keep training models until and unless a lower error is received.



```
Training:
    For all $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$, initialize $p_1^t = 1/N$
    For all base-learners $j = 1, \dots, L$
        Randomly draw $\mathcal{X}_j$ from $\mathcal{X}$ with probabilities $p_j^t$
        Train $d_j$ using $\mathcal{X}_j$
        For each $(x^t, r^t)$, calculate $y_j^t \leftarrow d_j(x^t)$
        Calculate error rate: $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$
        If $\epsilon_j > 1/2$, then $L \leftarrow j - 1$; stop
        $\beta_j \leftarrow \epsilon_j / (1 - \epsilon_j)$
        For each $(x^t, r^t)$, decrease probabilities if correct:
            If $y_j^t = r^t$, then $p_{j+1}^t \leftarrow \beta_j p_j^t$ Else $p_{j+1}^t \leftarrow p_j^t$
        Normalize probabilities:
            $Z_j \leftarrow \sum_t p_{j+1}^t$;  $p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$
Testing:
    Given $x$, calculate $d_j(x), j = 1, \dots, L$
    Calculate class outputs, $i = 1, \dots, K$:
        $y_i = \sum_{j=1}^L \left(\log \frac{1}{\beta_j}\right) d_{ji}(x)$
```

Figure 17.2 AdaBoost algorithm.

Many variants of AdaBoost have been proposed; here, we discuss the original algorithm AdaBoost.M1 (see figure 17.2). The idea is to modify the probabilities of drawing the instances as a function of the error. Let us say p_j^t denotes the probability that the instance pair (x^t, r^t) is drawn to train the jth base-learner. Initially, all $p_1^t = 1/N$. Then we add new base-learners as follows, starting from $j = 1$: ϵ_j denotes the error rate of d_j. AdaBoost requires that learners are weak, that is, $\epsilon_j < 1/2, \forall j$; if not, we stop adding new base-learners. Note that this error rate is not on the original problem but on the dataset used at step j. We define $\beta_j = \epsilon_j/(1 - \epsilon_j) < 1$, and we set $p_{j+1}^t = \beta_j p_j^t$ if d_j correctly classifies x^t; otherwise, $p_{j+1}^t = p_j^t$. Because p_{j+1}^t should be probabilities, there is a normalization where we divide p_{j+1}^t by $\sum_t p_{j+1}^t$, so that they sum up to 1. This has the effect that the probability of a correctly classified instance is decreased, and the probability of a misclassified instance increases. Then a new sample of the same size is drawn from the original sample according to these modified probabilities, p_{j+1}^t, with replacement, and is used to train d_{j+1}.

This has the effect that d_{j+1} focuses more on instances misclassified by d_j; that is why the base-learners are chosen to be simple and not accurate, since otherwise the next training sample would contain only a few outlier and noisy instances repeated many times over. For example, with decision trees, *decision stumps*, which are trees grown only one or two levels, are used. So it is clear that these would have bias but the decrease in variance is larger and the overall error decreases. An algorithm like the linear discriminant has low variance, and we cannot gain by AdaBoosting linear discriminants.

Once training is done, AdaBoost is a voting method. Given an instance, all d_j decide and a weighted vote is taken where weights are proportional to the base-learners' accuracies (on the training set): $w_j = \log(1/\beta_j)$. Freund and Schapire (1996) showed improved accuracy in twenty-two benchmark problems, equal accuracy in one problem, and worse accuracy in four problems.

Advantages of a Boosting Model

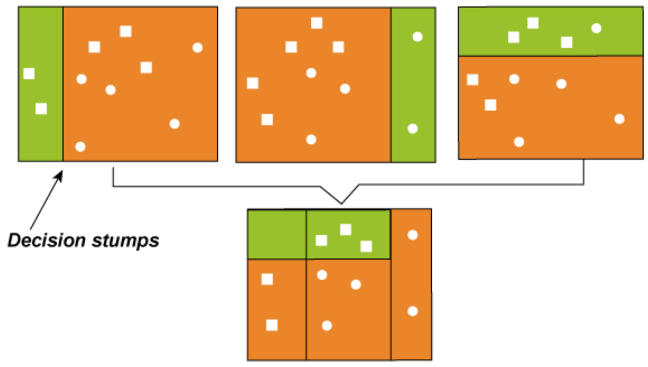
1. Boosting is a resilient method that curbs over-fitting easily.
2. Provably effective
3. Versatile — can be applied to a wide variety of problems.

Disadvantages of a Boosting Model

1. A disadvantage of boosting is that it is sensitive to outliers since every classifier is obliged to fix the errors in the predecessors. Thus, the method is too dependent on outliers.
2. Another disadvantage is that the method is almost impossible to scale up. This is because every estimator bases its correctness on the previous predictors, thus making the procedure difficult to streamline.

How does the Boosting Algorithm Work?

The basic principle behind the working of the boosting algorithm is to generate multiple weak learners and combine their predictions to form one strict rule. These weak rules are generated by applying base Machine Learning algorithms on different distributions of the data set. These algorithms generate weak rules for each iteration. After multiple iterations, the weak learners are combined to form a strong learner that will predict a more accurate outcome.



Unsupervised Learning: -

Unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as:

Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to **find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format**.

Types of Unsupervised Learning Algorithm:

The unsupervised learning algorithm can be further categorized into two types of problems:

1. **Clustering**
2. **Association**

Clustering in Machine Learning

Clustering or cluster analysis is a machine learning technique, which groups the unlabelled dataset.

It can be defined as ***"A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group."***

It does it by finding some similar patterns in the unlabelled dataset such as shape, size, color, behavior, etc., and divides them as per the presence and absence of those similar patterns.

It is an unsupervised learning method, hence no supervision is provided to the algorithm, and it deals with the unlabeled dataset.

After applying this clustering technique, each cluster or group is provided with a cluster-ID. ML system can use this id to simplify the processing of large and complex datasets.

The clustering technique is commonly used for **statistical data analysis.**

NOTE: - Clustering is somewhere similar to the classification algorithm, but the difference is the type of dataset that we are using. In classification, we work with the labeled data set, whereas in clustering, we work with the unlabeled dataset.

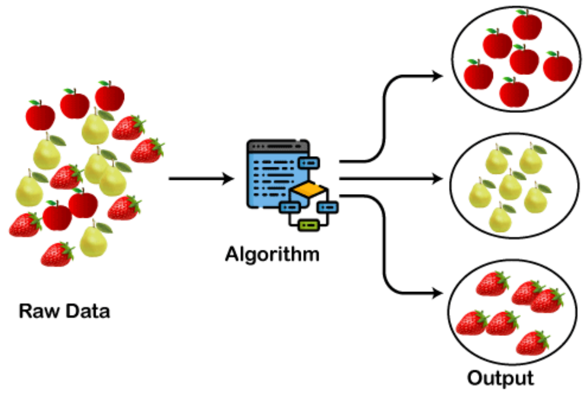
Example: Let's understand the clustering technique with the real-world example of Mall: When we visit any shopping mall, we can observe that the things with similar usage are grouped together. Such as the t-shirts are grouped in one section, and trousers are at other sections, similarly, at vegetable sections, apples, bananas, Mangoes, etc., are grouped in separate sections, so that we can easily find out the things. The clustering technique also works in the same way. Other examples of clustering are grouping documents according to the topic.

The clustering technique can be widely used in various tasks. Some most common uses of this technique are:

- Market Segmentation
- Statistical data analysis
- Social network analysis
- Image segmentation
- Anomaly detection, etc.

Apart from these general usages, it is used by the **Amazon** in its recommendation system to provide the recommendations as per the past search of products. **Netflix** also uses this technique to recommend the movies and web-series to its users as per the watch history.

The below diagram explains the working of the clustering algorithm. We can see the different fruits are divided into several groups with similar properties.



Types of Clustering Methods
Clustering itself can be categorized into two types viz. Hard Clustering and Soft Clustering. In hard clustering, one data point can belong to one cluster only. But in soft clustering, the output provided is a probability likelihood of a data point belonging to each of the pre-defined numbers of clusters. But there are also other various approaches of Clustering exist. Below are the main clustering methods used in Machine learning:

1. **Partitioning Clustering**
2. **Density-Based Clustering**
3. **Distribution Model-Based Clustering**
4. **Hierarchical Clustering**
5. **Fuzzy Clustering**

Partitioning Clustering: -
This method is one of the most popular choices for analysts to create clusters. In partitioning clustering, the clusters are partitioned based upon the characteristics of the data points. We need to specify the number of clusters to be created for this clustering method. These clustering algorithms follow an iterative process to reassign the data points between clusters based upon the distance. The algorithms that fall into this category are as follows

1. K-Means Clustering
2. PAM (Partitioning Around Medoids)
3. CLARA (Clustering Large Applications)

K-Means Clustering: -

- A K-means clustering algorithm tries to group similar items in the form of clusters. The number of groups is represented by K.
- K-means clustering uses the euclidean distance method to find out the distance between the points.
- K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.
- It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.
- It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Let us say we have an image that is stored with 24 bits/pixel and can have up to 16 million colors. Assume we have a color screen with 8 bits/pixel that can display only 256 colors. We want to find the best 256 colors among all 16 million colors such that the image using only the 256 colors in the palette looks as close as possible to the original image. This is color quantization where we map from high to lower resolution. In the general case, the aim is to map from a continuous space to a discrete space; this process is called vector quantization.
Of course we can always quantize uniformly, but this wastes the colormap by assigning entries to colors not existing in the image, or would not assign extra entries to colors frequently used in the image. For example, if the image is a seascape, we expect to see many shades of blue and maybe no red. So the distribution of the colormap entries should reflect the original density as close as possible placing many entries in high-density regions, discarding regions where there is no data.

Let us say we have a sample of $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$. We have k *reference vectors*, $\boldsymbol{m}_j, j = 1, \ldots, k$. In our example of color quantization, $\boldsymbol{x}^t$ are the image pixel values in 24 bits and $\boldsymbol{m}_j$ are the color map entries also in 24 bits, with $k = 256$.

Assume for now that we somehow have the $\boldsymbol{m}_j$ values; we will discuss how to learn them shortly. Then in displaying the image, given the pixel, $\boldsymbol{x}^t$, we represent it with the most similar entry, $\boldsymbol{m}_i$ in the color map, satisfying

$$\|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\|$$

That is, instead of the original data value, we use the closest value we have in the alphabet of reference vectors. $\boldsymbol{m}_i$ are also called *codebook vectors* or *code words*, because this is a process of *encoding/decoding* (see figure 7.1): Going from $\boldsymbol{x}^t$ to i is a process of encoding the data using the codebook of $\boldsymbol{m}_i, i = 1, \ldots, k$ and, on the receiving end, generating $\boldsymbol{m}_i$ from i is decoding. Quantization also allows *compression*: For example, instead of using 24 bits to store (or transfer over a communication line) each $\boldsymbol{x}^t$, we can just store/transfer its index i in the colormap using 8 bits to index any one of 256, and we get a compression rate of almost 3; there is also the color map to store/transfer.

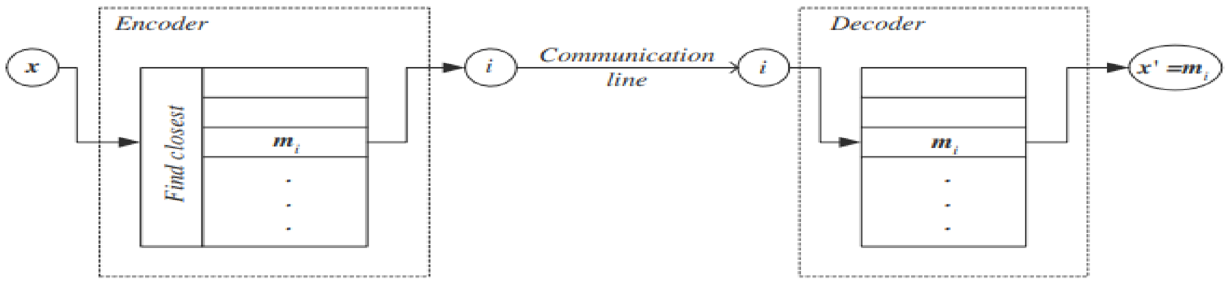


Figure 7.1 Given $\boldsymbol{x}$, the encoder sends the index of the closest code word and the decoder generates the code word with the received index as $\boldsymbol{x}'$. Error is $\|\boldsymbol{x}' - \boldsymbol{x}\|^2$.

Let us see how we can calculate $\boldsymbol{m}_i$: When $\boldsymbol{x}^t$ is represented by $\boldsymbol{m}_i$, there is an error that is proportional to the distance, $\|\boldsymbol{x}^t - \boldsymbol{m}_i\|$. For the new image to look like the original image, we should have these distances as small as possible for all pixels. The total *reconstruction error* is defined as

$$(7.3) \qquad E(\{\boldsymbol{m}_i\}_{i=1}^k | \mathcal{X}) = \sum_t \sum_i b_i^t \|\boldsymbol{x}^t - \boldsymbol{m}_i\|^2$$

where

$$(7.4) \qquad b_i^t = \begin{cases} 1 & \text{if } \|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\| \\ 0 & \text{otherwise} \end{cases}$$

The best reference vectors are those that minimize the total reconstruction error. b_i^t also depend on $\boldsymbol{m}_i$, and we cannot solve this optimization problem analytically. We have an iterative procedure named *k-means clustering* for this: First, we start with some $\boldsymbol{m}_i$ initialized randomly. Then at each iteration, we first use equation 7.4 and calculate b_i^t for all $\boldsymbol{x}^t$, which are the *estimated labels*; if b_i^t is 1, we say that $\boldsymbol{x}^t$ belongs to the group of $\boldsymbol{m}_i$. Then, once we have these labels, we minimize equation 7.3. Taking its derivative with respect to $\boldsymbol{m}_i$ and setting it to 0, we get

$$\boldsymbol{m}_i = \frac{\sum_t b_i^t \boldsymbol{x}^t}{\sum_t b_i^t}$$

The reference vector is set to the mean of all the instances that it represents. Note that this is the same as the formula for the mean in equation 7.2, except that we place the estimated labels b_i^t in place of the labels r_i^t. This is an iterative procedure because once we calculate the new $\boldsymbol{m}_i$, b_i^t change and need to be recalculated, which in turn affect $\boldsymbol{m}_i$. These two steps are repeated until $\boldsymbol{m}_i$ stabilize (see figure 7.2). The pseudocode of the k-means algorithm is given in figure 7.3.

One disadvantage is that this is a local search procedure, and the final $\boldsymbol{m}_i$ highly depend on the initial $\boldsymbol{m}_i$. There are various methods for initialization:

- One can simply take randomly selected k instances as the initial $\boldsymbol{m}_i$.
- The mean of all data can be calculated and small random vectors may be added to the mean to get the k initial $\boldsymbol{m}_i$.
- One can calculate the principal component, divide its range into k equal intervals, partitioning the data into k groups, and then take the means of these groups as the initial centers.

How does the K-Means Algorithm Work?

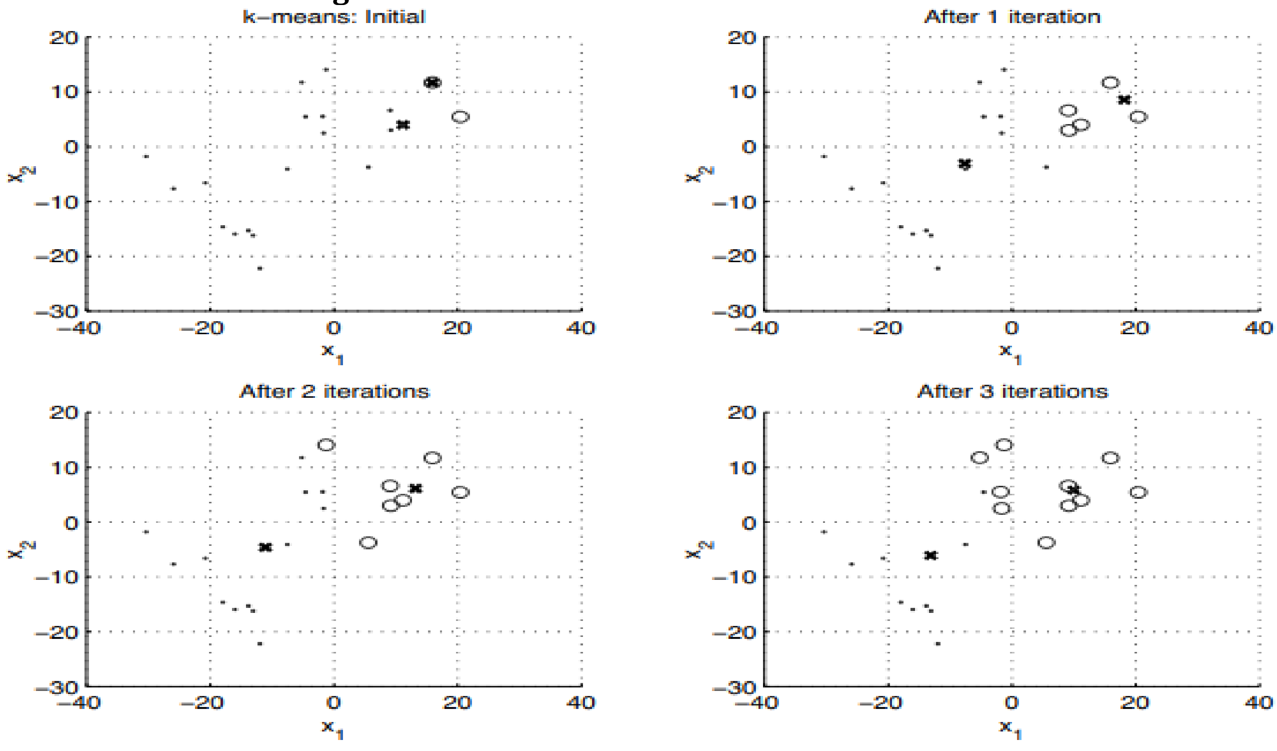


Figure 7.2 Evolution of k-means. Crosses indicate center positions. Data points are marked depending on the closest center.

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which mean reassign each data point to the new closest centroid of each cluster.
Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.
Step-7: The model is ready.

```
Initialize $\boldsymbol{m}_i, i = 1, \ldots, k$, for example, to $k$ random $\boldsymbol{x}^t$
Repeat
    For all $\boldsymbol{x}^t \in \mathcal{X}$
        $b_i^t \leftarrow \begin{cases} 1 & \text{if } \|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\| \\ 0 & \text{otherwise} \end{cases}$
    For all $\boldsymbol{m}_i, i = 1, \ldots, k$
        $\boldsymbol{m}_i \leftarrow \sum_t b_i^t \boldsymbol{x}^t / \sum_t b_i^t$
Until $\boldsymbol{m}_i$ converge
```

Figure 7.3 *k*-means algorithm.

How to choose the value of "K number of clusters" in K-means Clustering?

There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K.

- **The method is given below:**
- **Elbow Method**

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below

$$\text{WCSS} = \sum_{P_i \text{ in Cluster1}} \text{distance}(P_i\ C_1)^2 + \sum_{P_i \text{ in Cluster2}} \text{distance}(P_i\ C_2)^2 + \sum_{P_i \text{ in CLuster3}} \text{distance}(P_i\ C_3)^2$$

In the above formula of WCSS,

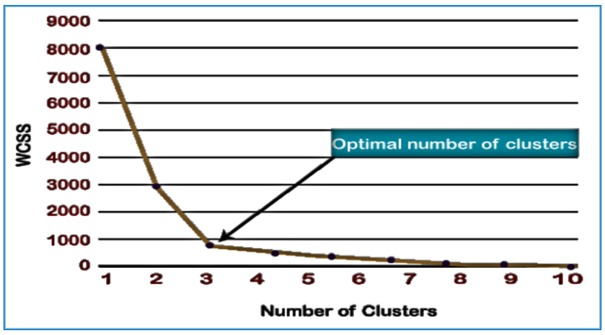
$\sum_{P_i \text{ in Cluster1}} \text{distance}(P_i\ C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



Note: We can choose the number of clusters equal to the given data points. If we choose the number of clusters equal to the data points, then the value of WCSS becomes zero, and that will be the endpoint of the plot

Expectation-Maximization Algorithm: -

The EM algorithm is considered a latent variable model to find the local maximum likelihood parameters of a statistical model, proposed by Arthur Dempster, Nan Laird, and Donald Rubin in 1977.

The EM (Expectation-Maximization) algorithm is one of the most commonly used terms in machine learning to obtain maximum likelihood estimates of variables that are sometimes observable and sometimes not. However, it is also applicable to unobserved data or sometimes called latent. It has various real-world applications in statistics, including obtaining the **mode of the posterior marginal distribution of parameters in machine learning and data mining applications**.

In most real-life applications of machine learning, it is found that several relevant learning features are available, but very few of them are observable, and the rest are unobservable. If the variables are observable, then it can predict the value using instances.

On the other hand, the variables which are latent or directly not observable, for such variables Expectation-Maximization (EM) algorithm plays a vital role to predict the value with the condition that the general form of probability distribution governing those latent variables is known to us. In this topic, we will discuss a basic introduction to the EM algorithm, a flow chart of the EM algorithm, its applications, advantages, and disadvantages of EM algorithm, etc.

What is an EM algorithm?

The Expectation-Maximization (EM) algorithm is defined as the combination of various unsupervised machine learning algorithms, which is used to determine the **local maximum likelihood estimates (MLE)** or **maximum a posteriori estimates (MAP)** for unobservable variables in statistical models. Further, it is a technique to find maximum likelihood estimation when the latent variables are present. It is also referred to as the **latent variable model.**

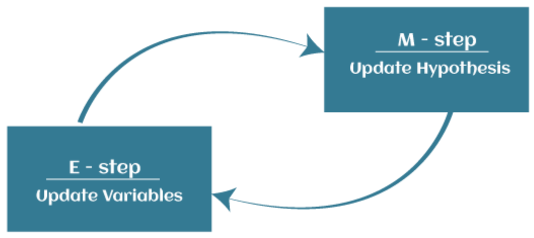
A latent variable model consists of both observable and unobservable variables where observable can be predicted while unobserved are inferred from the observed variable. These unobservable variables are known as latent variables.

Key Points:

- It is known as the latent variable model to determine MLE and MAP parameters for latent variables.
- It is used to predict values of parameters in instances where data is missing or unobservable for learning, and this is done until convergence of the values occurs.

EM Algorithm

The EM algorithm is the combination of various unsupervised ML algorithms, such as the **k-means clustering algorithm**. Being an iterative approach, it consists of two modes. In the first mode, we estimate the missing or latent variables. Hence it is referred to as the **Expectation/estimation step (E-step)**. Further, the other mode is used to optimize the parameters of the models so that it can explain the data more clearly. The second mode is known as the **maximization-step or M-step.**



- **Expectation step (E - step):** It involves the estimation (guess) of all missing values in the dataset so that after completing this step, there should not be any missing value.
- **Maximization step (M - step):** This step involves the use of estimated data in the E-step and updating the parameters.
- **Repeat E-**step and M-step until the convergence of the values occurs.

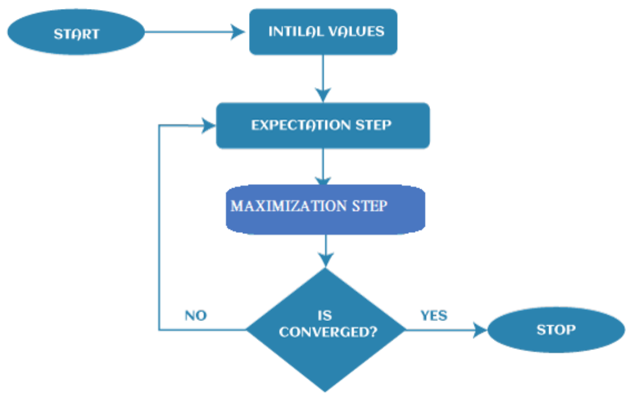
The primary goal of the EM algorithm is to use the available observed data of the dataset to estimate the missing data of the latent variables and then use that data to update the values of the parameters in the M-step.

What is Convergence in the EM algorithm?

Convergence is defined as the specific situation in probability based on intuition, e.g., if there are two random variables that have very less difference in their probability, then they are known as converged. In other words, whenever the values of given variables are matched with each other, it is called convergence.

Steps in EM Algorithm

The EM algorithm is completed mainly in 4 steps, which include I***nitialization Step, Expectation Step, Maximization Step, and convergence Step***. These steps are explained as follows:



- **1st Step:** The very first step is to initialize the parameter values. Further, the system is provided with incomplete observed data with the assumption that data is obtained from a specific model.
- **2nd Step:** This step is known as Expectation or E-Step, which is used to estimate or guess the values of the missing or incomplete data using the observed data. Further, E-step primarily updates the variables.
- **3rd Step:** This step is known as Maximization or M-step, where we use complete data obtained from the 2nd step to update the parameter values. Further, M-step primarily updates the hypothesis.
- **4th step:** The last step is to check if the values of latent variables are converging or not. If it gets "yes", then stop the process; else, repeat the process from step 2 until the convergence occurs.

Applications of EM algorithm

The primary aim of the EM algorithm is to estimate the missing data in the latent variables through observed data in datasets. The EM algorithm or latent variable model has a broad range of real-life applications in machine learning. These are as follows:

- The EM algorithm is applicable in data clustering in machine learning.

- It is often used in computer vision and NLP (Natural language processing).
- It is used to estimate the value of the parameter in mixed models such as the **Gaussian Mixture Model** and quantitative genetics.
- It is also used in psychometrics for estimating item parameters and latent abilities of item response theory models.
- It is also applicable in the medical and healthcare industry, such as in image reconstruction and structural engineering.
- It is used to determine the Gaussian density of a function.

Advantages of EM algorithm

- It is very easy to implement the first two basic steps of the EM algorithm in various machine learning problems, which are E-step and M- step.
- It is mostly guaranteed that likelihood will enhance after each iteration.
- It often generates a solution for the M-step in the closed form.

Disadvantages of EM algorithm

- The convergence of the EM algorithm is very slow.
- It can make convergence for the local optima only.
- It takes both forward and backward probability into consideration. It is opposite to that of numerical optimization, which takes only forward probabilities.

The importance of the EM algorithm can be seen in various applications:
data clustering, natural language processing (NLP), computer vision, image reconstruction, structural engineering, etc.

Supervised learning after clustering: -

Clustering, like the dimensionality reduction methods, can be used for two purposes: it can be used for data exploration, to understand the structure of data.

Dimensionality reduction methods are used to find correlations between variables and thus group variables; clustering methods, on the other hand, are used to find similarities between instances and thus group instances.

If such groups are found, these may be named (by application experts) and their attributes be defined. One can choose the group mean as the representative prototype of instances in the group, or the possible range of attributes can be written. This allows a simpler description of the data.

For example, if the customers of a company seem to fall in one of k groups, called segments, customers being defined in terms of their demographic attributes and transactions with the company, then a better understanding of the customer base will be provided that will allow the company to provide different strategies for different types of customers; this is part of customer relationship management (CRM). Likewise, the company will also be able to develop strategies for those customers who do not fall in any large group, and who may require attention, for example, churning customers.

Frequently, clustering is also used as a preprocessing stage. Just like the dimensionality reduction methods allowed us to make a mapping to a new space, after clustering, we also map to a new kdimensional space where the dimensions are hi (or bi at the risk of loss of information). In a supervised setting, we can then learn the discriminant or regression function in this new space. The difference from dimensionality reduction methods like PCA however is that k, the dimensionality of the new space, can be larger than d, the original dimensionality.

When we use a method like PCA, where the new dimensions are combinations of the original dimensions, to represent any instance in the new space, all dimensions contribute; that is, all z_j are nonzero. In the case of a method like clustering where the new dimensions are defined locally, there are many more new dimensions, b_j , but only one (or if we use h_j , few) of them have a nonzero value. In the former case, where there are few dimensions but all contribute, we have a distributed representation; in the latter case, where there are many dimensions but few contribute, we have a local representation.

One advantage of preceding a supervised learner with unsupervised clustering or dimensionality reduction is that the latter does not need labeled data. Labeling the data is costly. We can use a large amount of unlabeled data for learning the cluster parameters and then use a smaller labeled data to learn the second stage of classification or regression. Unsupervised learning is called "learning what normally happens" (Barrow 1989). When followed by a supervised learner, we first learn what normally happens and then learn what that means.

In the case of classification, when each class is a mixture model composed of a number of components, the whole density is a mixture of mixtures:

$$p(\mathbf{x}|C_i) = \sum_{j=1}^{k_i} p(\mathbf{x}|\mathcal{G}_{ij})P(\mathcal{G}_{ij})$$

$$p(\mathbf{x}) = \sum_{i=1}^{K} p(\mathbf{x}|C_i)P(C_i)$$

Where k_i is the number of components making up $p(x|C_i)$ and G_{ij} is the component j of class i. Note that different classes may need different number of components.

Spectral clustering: -

As the number of dimensions increases, a distance-based similarity measure converges to a constant value between any given examples. Reduce dimensionality either by using **PCA** on the feature data, or by using "spectral clustering" to modify the clustering algorithm as explained below.

Curse of Dimensionality and Spectral Clustering

These plots show how the ratio of the standard deviation to the mean of distance between examples decreases as the number of dimensions increases. This convergence means k-means becomes less effective at distinguishing between examples. This negative consequence of high-dimensional data is called the curse of dimensionality.

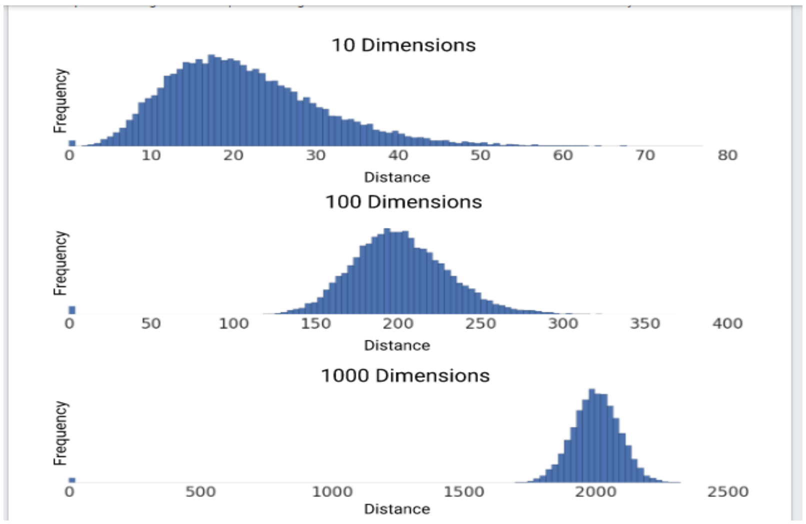
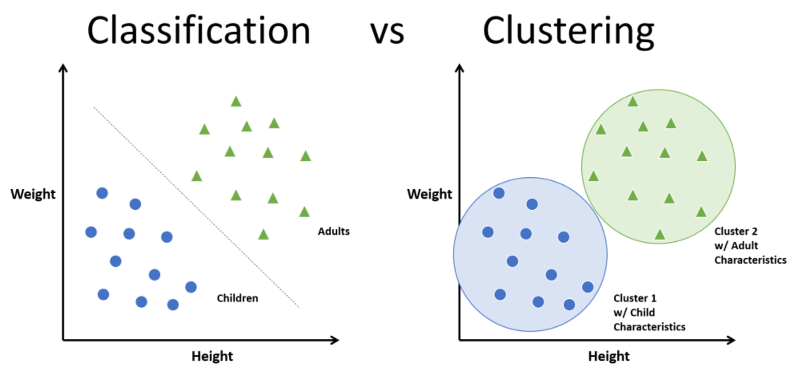


Figure 3: A demonstration of the curse of dimensionality. Each plot shows the pairwise distances between 200 random points.

Spectral clustering avoids the curse of dimensionality by adding a pre-clustering step to your algorithm:

- Reduce the dimensionality of feature data by using PCA.
- Project all data points into the lower-dimensional subspace.
- Cluster the data in this subspace by using your chosen algorithm.

Therefore, spectral clustering is not a separate clustering algorithm but a pre- clustering step that you can use with any clustering algorithm.

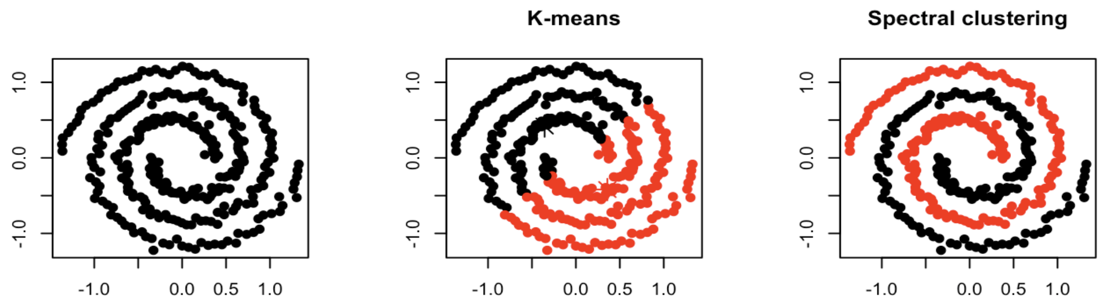


There are two major approaches in clustering. They are:

1. Compactness
2. Connectivity

In compactness, the points are closer to each other and are compact towards the cluster center. Distance is used as a measure to compute closeness. There are different types of distance metrics that are in use. A few of them are Euclidean distance, Manhattan distance, Minkowski distance, and Hamming distance.
K-means algorithm uses the compactness approach.

In connectivity, the points in a cluster are either immediately next to each other (epsilon distance) or connected. Even if the distance is less, they are not put in the same cluster.
Spectral clustering is one of the techniques to follow this approach.



How to do Spectral Clustering?

The three major steps involved in spectral clustering are: constructing a similarity graph, projecting data onto a lower-dimensional space, and clustering the data. Given a set of points S in a higher-dimensional space, it can be elaborated as follows:

1. Form a distance matrix
2. Transform the distance matrix into an affinity matrix A
3. Compute the degree matrix D and the Laplacian matrix L = D – A.
4. Find the eigenvalues and eigenvectors of L.
5. With the eigenvectors of k largest eigenvalues computed from the previous step form a matrix.
6. Normalize the vectors.
7. Cluster the data points in k-dimensional space.

Building the Similarity Graph: This step builds the Similarity Graph in the form of an adjacency matrix which is represented by A.

Projecting the data onto a lower Dimensional Space: This step is done to account for the possibility that members of the same cluster may be far away in the given dimensional space. Thus the dimensional space is reduced so that those points are closer in the reduced dimensional space and thus can be clustered together by a traditional clustering algorithm. It is done by computing the **Graph Laplacian Matrix**. To compute it though first, the degree of a node needs to be defined.

The degree of the ith node is given by $d_i = \sum_{j=1|(i,j)\epsilon E}^{n} w_{ij}$ Note that w_{ij} is the edge between the nodes i and j as defined in the adjacency matrix above. The degree matrix is defined as follows:-

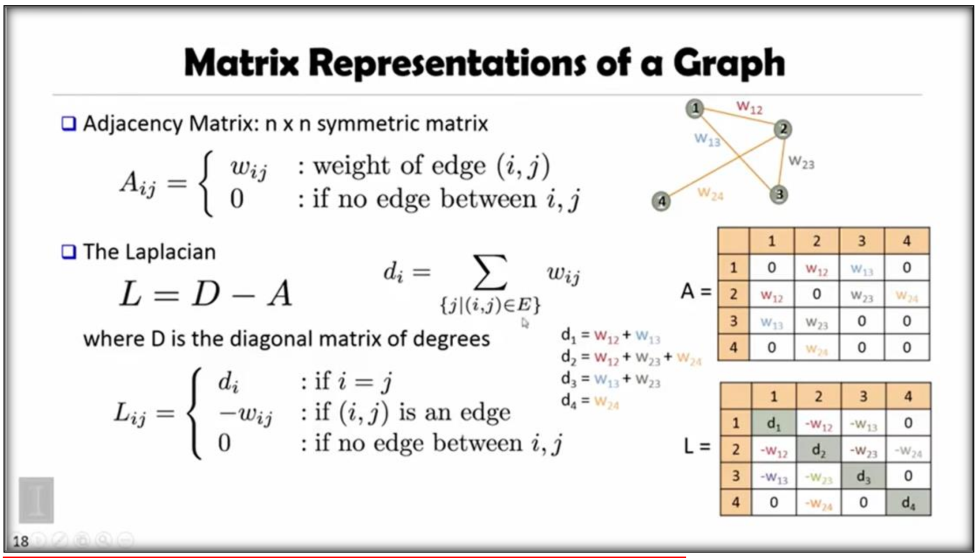
$$D_{ij} = \begin{cases} d_i, i = j \\ 0, i \neq j \end{cases}$$

Thus the Graph Laplacian Matrix is defined as:-

$$L = D - A$$

This Matrix is then normalized for mathematical efficiency. To reduce the dimensions, first, the eigenvalues and the respective eigenvectors are calculated. If the number of clusters is k then the first eigenvalues and their eigen-vectors are taken and stacked into a matrix such that the eigen-vectors are the columns.

Clustering the Data: - This process mainly involves clustering the reduced data by using any traditional clustering technique – typically K-Means Clustering. First, each node is assigned a row of the normalized of the Graph Laplacian Matrix. Then this data is clustered using any traditional technique. To transform the clustering result, the node identifier is retained.



Matrix Representations of a Graph

- Adjacency Matrix: n x n symmetric matrix

$$A_{ij} = \begin{cases} w_{ij} & : \text{weight of edge } (i,j) \\ 0 & : \text{if no edge between } i,j \end{cases}$$

- The Laplacian

$$L = D - A \qquad d_i = \sum_{\{j|(i,j)\in E\}} w_{ij}$$

where D is the diagonal matrix of degrees

$$L_{ij} = \begin{cases} d_i & : \text{if } i = j \\ -w_{ij} & : \text{if } (i,j) \text{ is an edge} \\ 0 & : \text{if no edge between } i,j \end{cases}$$

$d_1 = w_{12} + w_{13}$
$d_2 = w_{12} + w_{23} + w_{24}$
$d_3 = w_{13} + w_{23}$
$d_4 = w_{24}$

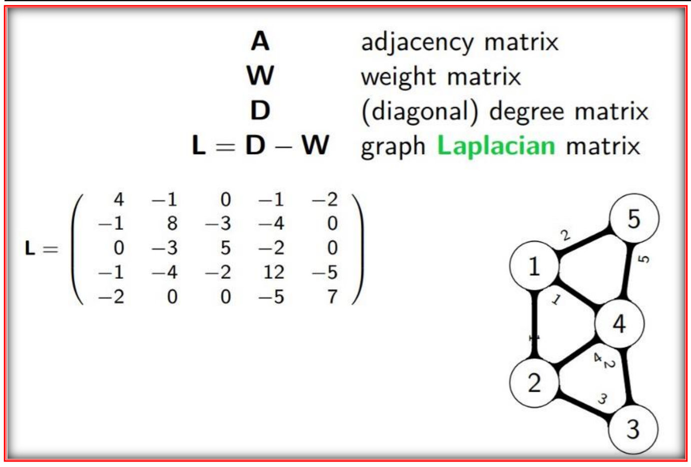
A =

	1	2	3	4
1	0	w_{12}	w_{13}	0
2	w_{12}	0	w_{23}	w_{24}
3	w_{13}	w_{23}	0	0
4	0	w_{24}	0	0

L =

	1	2	3	4
1	d_1	$-w_{12}$	$-w_{13}$	0
2	$-w_{12}$	d_2	$-w_{23}$	$-w_{24}$
3	$-w_{13}$	$-w_{23}$	d_3	0
4	0	$-w_{24}$	0	d_4

18



A	adjacency matrix
W	weight matrix
D	(diagonal) degree matrix
L = D − W	graph Laplacian matrix

$$\mathbf{L} = \begin{pmatrix} 4 & -1 & 0 & -1 & -2 \\ -1 & 8 & -3 & -4 & 0 \\ 0 & -3 & 5 & -2 & 0 \\ -1 & -4 & -2 & 12 & -5 \\ -2 & 0 & 0 & -5 & 7 \end{pmatrix}$$

Pros and Cons of Spectral Clustering

Spectral clustering helps us overcome two major problems in clustering: one being the shape of the cluster and the other is determining the cluster centroid. K-means algorithm generally assumes that the clusters are spherical or round i.e. within k-radius from the cluster centroid. In K means, much iteration is required to determine the cluster centroid. In spectral, the clusters do not follow a fixed shape or pattern. Points that are far away but connected belong to the same cluster and the points which are less distant from each other could belong to different clusters if they are not connected. This implies that the algorithm could be effective for data of different shapes and sizes.

Where can I use spectral clustering?
Spectral clustering has its application in many areas which includes:
Image segmentation, educational data mining, entity resolution, speech separation, spectral clustering of protein sequences, text image segmentation.
Though spectral clustering is a technique based on graph theory, the approach is used to identify communities of vertices in a graph based on the edges connecting them. This method is flexible and allows us to cluster non-graph data as well either with or without the original data.

Choosing number of clusters: -
Like any learning method, clustering also has its knob to adjust complexity; it is k, the number of clusters. Given any k, clustering will always find k centers, whether they really are meaningful groups, or whether they are imposed by the method we use. There are various ways we can use to fine-tune k:

- In some applications such as color quantization, k is defined by the application.
- Plotting the data in two dimensions using PCA may be used in uncovering the structure of data and the number of clusters in the data.
- An incremental approach may also help: setting a maximum allowed distance is equivalent to setting a maximum allowed reconstruction error per instance.
- In some applications, validation of the groups can be done manually by checking whether clusters actually code meaningful groups of the data. For example, in a data mining application, application experts may do this check. In color quantization, we may inspect the image visually to check its quality (despite the fact that our eyes and brain do not analyze an image pixel by pixel).

Depending on what type of clustering method we use, we can plot the reconstruction error or log likelihood as a function of k and look for the "elbow." After a large enough k, the algorithm will start dividing groups, in which case there will not be a large decrease in the reconstruction error or large increase in the log likelihood. Similarly, in hierarchical clustering, by looking at the differences between levels in the tree, we can decide on a good split.

UNIT WISE IMPORTANT QUESTIONS: -

1. List and explain different ways the multiple base learners arecombined to generate the final output.
2. What is Voting and How Does it Work? Explain it with Examples.
3. What is Ensemble modeling? Discuss about Bagging and Boosting.
4. What is Boosting and How Does it Work? Explain it with examples.
5. What is bagging and How Does it Work? Explain it with examples.
6. What is the Difference between Bagging and Boosting? Which isbetter?
7. How does the AdaBoost algorithm work?
8. Make use of example Illustrate K means clustering algorithm.
9. Use K Means clustering to cluster the following data into twogroups. Assume cluster centroid are $m_1=2$ and $m_2=4$. The distance function used is Euclidean distance. { 2, 4, 10, 12, 3, 20, 30, 11, 25}
10. Explain the steps involved in expectation maximization algorithm
11. What is the need of supervised learning after clustering
12. With an example explain spectral clustering.
13. Identify the various ways of choosing number of clusters.
14. Let us say we have three classification algorithms. How can we order these three from best to worst?
15. Propose a suitable test to compare the errors of two regression algorithms.
16. If we have two variants of algorithm A and three variants of algorithm B, how can we compare the overall accuracies of A and B taking all their variants into account?
17. In image compression, k-means can be used as follows: The image is divided into no overlapping c×c windows and these c2-dimensional vectors make up the sample. For a given k, which is generally a

power of two, we do k-means clustering. The reference vectors and the indices for each window is sent over the communication line. At the receiving end, the image is then reconstructed by reading from the table of reference vectors using the indices. Write the computer program that does this for different values of k and c. For each case, calculate the reconstruction error and the compression rate.

Answer: -

```
Initialize $\boldsymbol{m}_i, i = 1, \ldots, k$, for example, to $k$ random $\boldsymbol{x}^t$
Repeat
    For all $\boldsymbol{x}^t \in \mathcal{X}$
        $b_i^t \leftarrow \begin{cases} 1 & \text{if } \|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\| \\ 0 & \text{otherwise} \end{cases}$
    For all $\boldsymbol{m}_i, i = 1, \ldots, k$
        $\boldsymbol{m}_i \leftarrow \sum_t b_i^t \boldsymbol{x}^t / \sum_t b_i^t$
Until $\boldsymbol{m}_i$ converge
```

Figure 7.3 *k*-means algorithm.

18. How can we make k-means robust to outliers?

Answer: -

This can be done by solving problems of the outlier's sensitivity. We can achieve the robust for outliers. This will be done by evaluating each object, and counting the number of its neighbors, eventually some objects will look like sole or isolated objects, that mean that the closest neighbor is further than it should be. The second technique will alter the way of measuring distances in K-means by adding some techniques inspired from the ROCK algorithm, the rock algorithm depends on a new concept of distance, it does not depend on the traditional concept of K-means, it counts the number of neighbors between two points, the more the number the strongest the relation and the similarity, that concept will be make K-means able to discover non-globular shapes.