SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Machine Learning Techniques – SEC1630

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT I – INTRODUCTION TO MACHINE LEARNING

UNIT I
INTRODUCTION TO MACHINE LEARNING

Machine Learning vs Statistical Modelling, Applications of Machine Learning, Supervised vs Unsupervised Learning, Supervised Learning Classification, Unsupervised Learning Classification, Python libraries suitable for Machine Learning.

Definition of Machine learning:

Well posed learning problem: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."(Tom Michel)

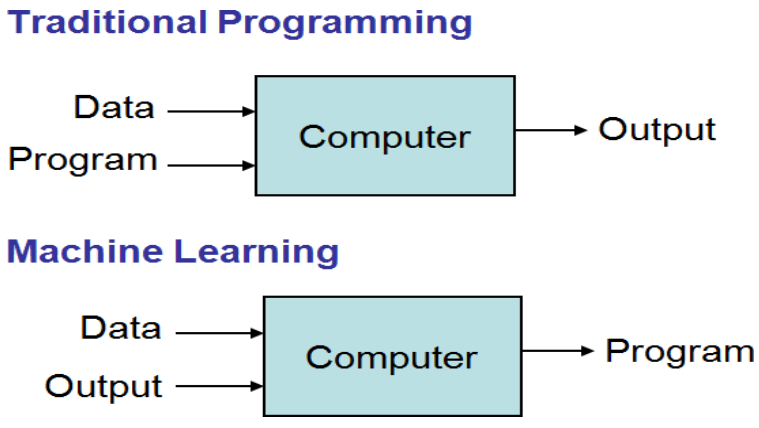
"Field of study that gives computers the ability to learn without being explicitly programmed".

Learning = Improving with experience at some task

- Improve over task T,
- with respect to performance measure P,
- based on experience E.

E.g., Learn to play checkers

- T : Play checkers
- P : % of games won in world tournament
- E: opportunity to play against self



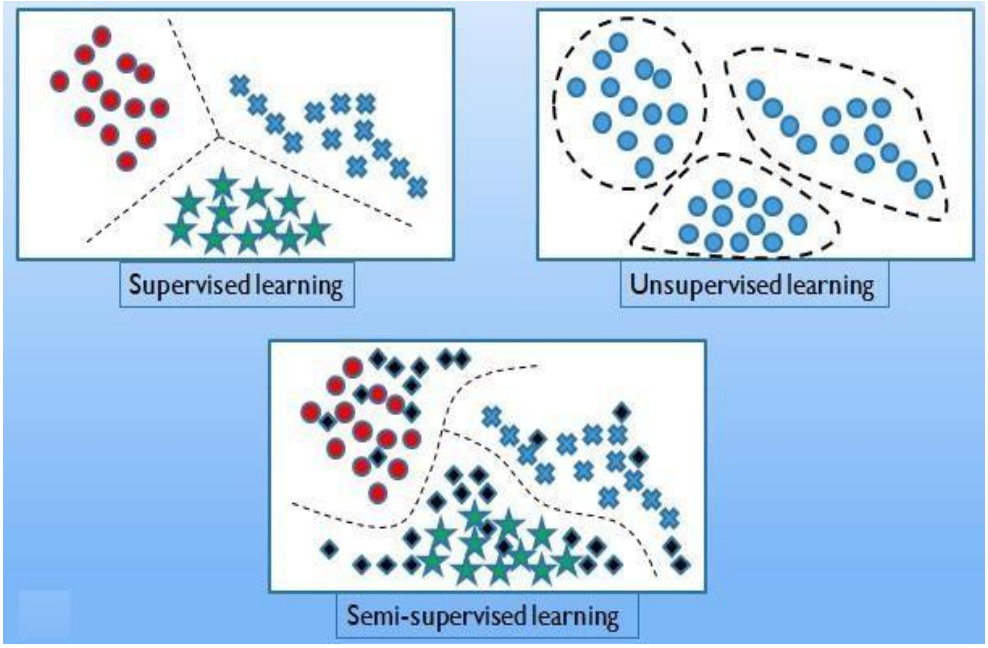
Examples of tasks that are best solved by using a learning algorithm:

- Recognizing patterns:
  - Facial identities or facial expressions
  - Handwritten or spoken words
  - Medical images
- Generating patterns:
  - Generating images or motion sequences (demo)
- Recognizing anomalies:
  - Unusual sequences of credit card transactions
  - Unusual patterns of sensor readings in a nuclear power plant or unusual sound in your car engine.
- Prediction:
  - Future stock prices or currency exchange rates
- The web contains a lot of data. Tasks with very big datasets often use machine learning
  - especially if the data is noisy or non-stationary.
- Spam filtering, fraud detection:
  - The enemy adapts so we must adapt too.
- Recommendation systems:
  - Lots of noisy data. Million dollar prize!
- Information retrieval:
  - Find documents or images with similar content.
- Data Visualization:
  - Display a huge database in a revealing way

Types of learning algorithms:

- Supervised learning
  - Training data includes desired outputs. Examples include,
    - Prediction
    - Classification (discrete labels), Regression (real values)

- Unsupervised learning
  - Training data does not include desired outputs, Examples include,
    - Clustering
    - Probability distribution estimation
    - Finding association (in features)
    - Dimension reduction
- Semi-supervised learning
  - Training data includes a few desired outputs
- Reinforcement learning
  - Rewards from sequence of actions
    - Policies: what actions should an agent take in a particular situation
    - Utility estimation: how good is a state (→used by policy)
  - No supervised output but delayed reward
  - Credit assignment problem (what was responsible for the outcome)
  - Applications:
    - Game playing
    - Robot in a maze
    - Multiple agents, partial observability, ...



Hypothesis Space

- One way to think about a supervised learning machine is as a device that explores a "hypothesis space".
  - Each setting of the parameters in the machine is a different hypothesis about the function that maps input vectors to output vectors.
  - If the data is noise-free, each training example rules out a region of hypothesis space.
  - If the data is noisy, each training example scales the posterior probability of each point in the hypothesis space in proportion to how likely the training example is given that hypothesis.
- The art of supervised machine learning is in:
  - Deciding how to represent the inputs and outputs
  - Selecting a hypothesis space that is powerful enough to represent the relationship between inputs and outputs but simple enough to be searched.

Generalization

- The real aim of supervised learning is to do well on test data that is not known during learning.
- Choosing the values for the parameters that minimize the loss function on the training data is not necessarily the best policy.
- We want the learning machine to model the true regularities in the data and to ignore the noise in the data.
  - But the learning machine does not know which regularities are real and which are accidental quirks of the particular set of training examples we happen to pick.
- So how can we be sure that the machine will generalize correctly to new data?

Training set, Test set and Validation set

- Divide the total dataset into three subsets:
  - Training data is used for learning the parameters of the model.
  - Validation data is not used of learning but is used for deciding what type of model and what amount of regularization works best.
  - Test data is used to get a final, unbiased estimate of how well the network works. We expect this estimate to be worse than on the validation data.

- We could then re-divide the total dataset to get another unbiased estimate of the true error rate.

Learning Associations:

- Basket analysis:

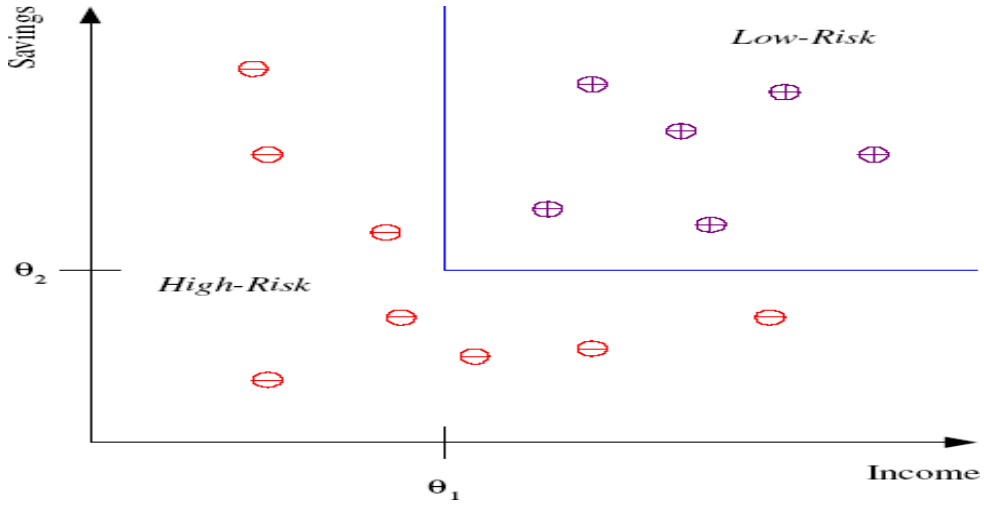
$P(Y \mid X)$ probability that somebody who buys X also buys Y where X and Y are products/services.

Example: P (chips | beer) = 0.7 // 70 percent of customers who buy beer also buy chips.

We may want to make a distinction among customers and toward this, estimate P(Y|X,D) where D is the set of customer attributes, for example, gender, age, marital status, and so on, assuming that we have access to this information. If this is a bookseller instead of a supermarket, products can be books or authors. In the case of a Web portal, items correspond to links to Web pages, and we can estimate the links a user is likely to click and use this information to download such pages in advance for faster access.

Classification:

- Example: Credit scoring
- Differentiating between low-risk and high-risk customers from their income and savings
- Discriminant: IF income > θ1 AND savings > θ2 THEN low-risk ELSE high-risk



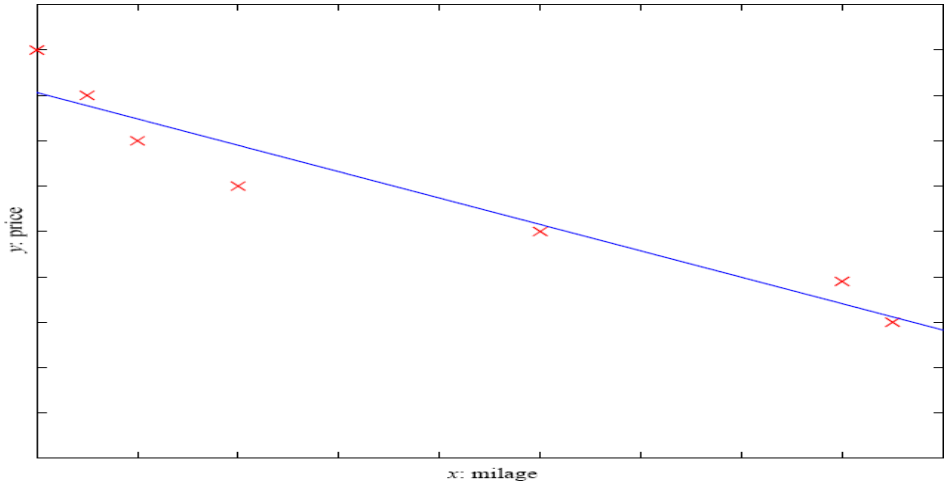
Prediction - Regression:

- Example: Predict Price of a used car
- x : car attributes

  y : price

  $y = g(x \mid \theta)$

  where, $g(\ )$ is the model, θ are the parameters



A training dataset of used cars and the function fitted. For simplicity, mileage is taken as the only input attribute and a linear model is used.

Supervised Learning:

Learning a Class from Examples:

- Class C of a "family car"
  - Prediction: Is car x a family car?
  - Knowledge extraction: What do people expect from a family car?
- Output:

  Positive (+) and negative (–) examples
- Input representation:

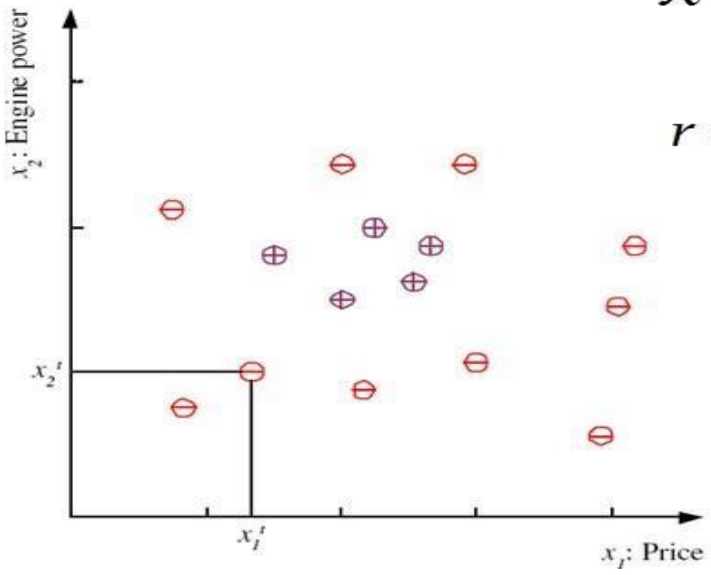
  x_1: price, x_2 : engine power

Training set X

$$\mathcal{X} = \{\boldsymbol{x}^t, r^t\}_{t=1}^N$$

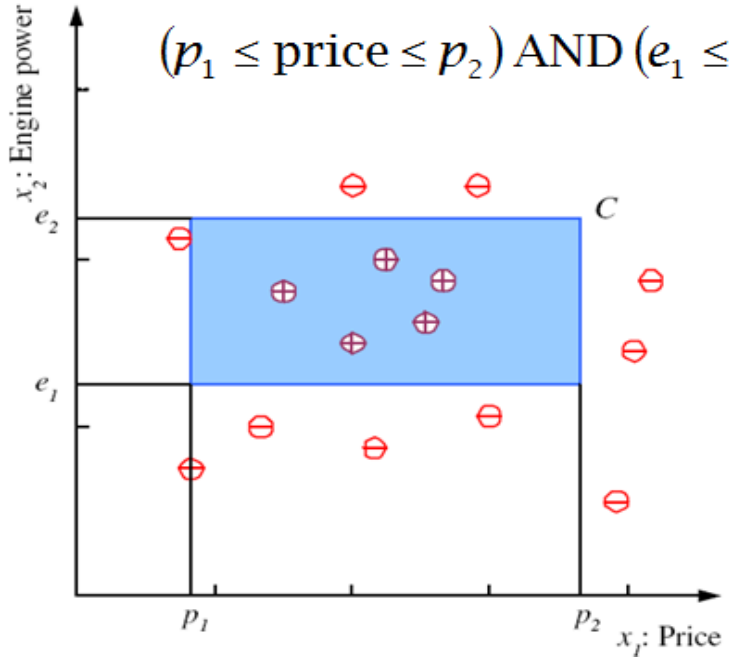
$$r = \begin{cases} 1 \text{ if } \boldsymbol{x} \text{ is positive} \\ 0 \text{ if } \boldsymbol{x} \text{ is negative} \end{cases}$$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$



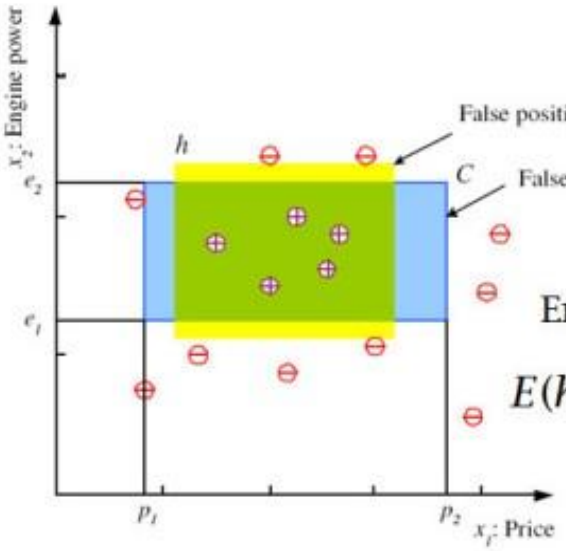
Class C

$$(p_1 \le \text{price} \le p_2) \text{ AND } (e_1 \le \text{engine power} \le e_2)$$



Hypothesis class $\mathcal{H}$

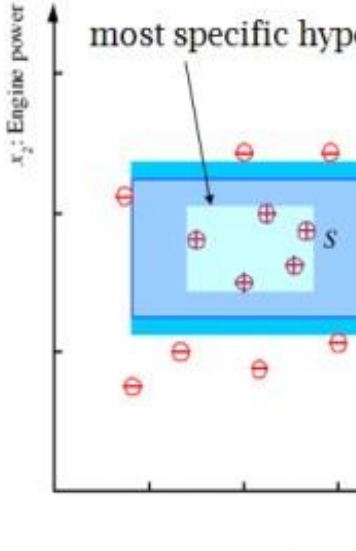
$$h(\boldsymbol{x}) = \begin{cases} 1 \text{ if } h \text{ classifies } \boldsymbol{x} \text{ as positive} \\ 0 \text{ if } h \text{ classifies } \boldsymbol{x} \text{ as negative} \end{cases}$$



Error of h on $\mathcal{H}$

$$E(h \mid \mathcal{X}) = \sum_{t=1}^{N} 1\left(h\left(\boldsymbol{x}^t\right) \neq r^t\right)$$

S, G, and the Version Space



$h \in \mathcal{H}$, between S and G is consistent

and make up the version space

Probably Approximately Correct (PAC):

- Cannot expect a learner to learn a concept exactly.
- Cannot always expect to learn a close approximation to the target concept
- Therefore, the only realistic expectation of a good learner is that with high probability it will learn a close approximation to the target concept.
- In Probably Approximately Correct (PAC) learning, one requires that given small parameters ε and δ, with probability at least (1- δ) a learner produces a hypothesis with error at most ε.
- The reason we can hope for that is the Consistent Distribution assumption.

PAC Learnability

- Consider a concept class C defined over an instance space X (containing instances of length n), and a learner L using a hypothesis space H.
- C is PAC learnable by L using H if
  - for all $f \in C$,
  - for all distributions D over X, and fixed $0< \varepsilon , \delta < 1$,
- L, given a collection of m examples sampled independently according to D produces
  - with probability at least (1- δ) a hypothesis $h \in H$ with error at most ε, ($Error_D = Pr_D[f(x) : = h(x)]$)
- where m is polynomial in 1/ ε, 1/ δ, n and size(H)
- C is efficiently learnable if L can produce the hypothesis in time polynomial in 1/ ε, 1/ δ, n and size(H)
- We impose two limitations:
- Polynomial sample complexity (information theoretic constraint)
  - Is there enough information in the sample to distinguish a hypothesis h that approximate f ?
- Polynomial time complexity (computational complexity)
  - Is there an efficient algorithm that can process the sample and produce a good hypothesis h ?

- To be PAC learnable, there must be a hypothesis h □ H with arbitrary small error for every f ∈ C. We generally assume H ⊇ C. (Properly PAC learnable if H=C)

Occam's Razor:

Claim: The probability that there exists a hypothesis h ∈ H that
(1) is consistent with m examples and
(2) satisfies error(h) > ε ($Error_D(h) = Pr_{x \in D}[f(x) \neg = h(x)]$)
is less than $|H|(1-\varepsilon)^m$.

Proof: Let h be such a bad hypothesis.
- The probability that h is consistent with one example of f is

$$\Pr_{x \in D}[f(x) = h(x)] < 1 - \varepsilon$$

- Since the m examples are drawn independently of each other,
The probability that h is consistent with m example of f is less than $(1-\varepsilon)^m$

- The probability that *some* hypothesis in H is consistent with m examples
is less than $|H|(1-\varepsilon)^m$

We want this probability to be smaller than δ, that is:

$$|H|(1-\varepsilon)^m < \delta$$

$$\ln(|H|) + m \ln(1-\varepsilon) < \ln(\delta)$$

(with $e^{-x} = 1-x+x^2/2+\ldots$; $e^{-x} > 1-x$; $\ln(1-\varepsilon) < -\varepsilon$; gives a safer δ)

$$m > \frac{1}{\varepsilon}\{\ln(|H|) + \ln(1/\delta)\}$$

(gross over estimate)
It is called Occam's razor, because it indicates a preference towards small hypothesis spaces

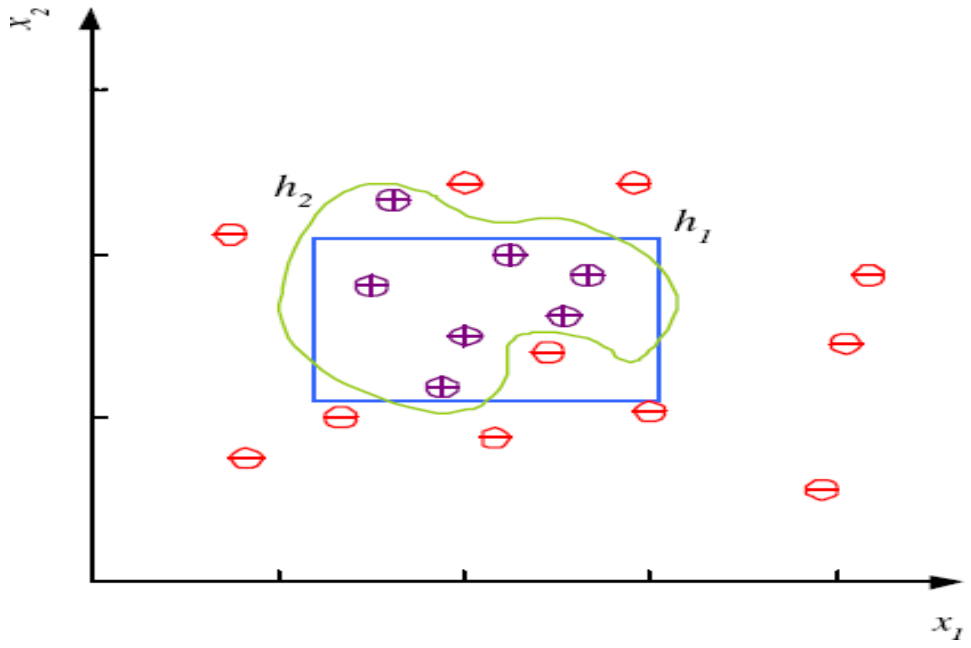
Noise and Model Complexity:

Noise is any unwanted anomaly in the data and due to noise, the class may be more difficult to learn and zero error may be infeasible with a simple hypothesis class. There are several interpretations of noise:

- There may be imprecision in recording the input attributes, which may shift the data points in the input space.
- There may be errors in labeling the data points, which may relabel positive instances as negative and vice versa. This is sometimes called teacher noise.
- There may be additional attributes, which we have not taken into account, that affect the label of an instance. Such attributes may be hidden or latent in that they may be unobservable. The effect of these neglected attributes is thus modeled as a random component and is included in "noise."

Use the simpler one because

- Simpler to use (lower computational complexity)
- Easier to train (lower space complexity)
- Easier to explain (more interpretable)
- Generalizes better (lower variance - Occam's razor)



Learning Multiple Classes:

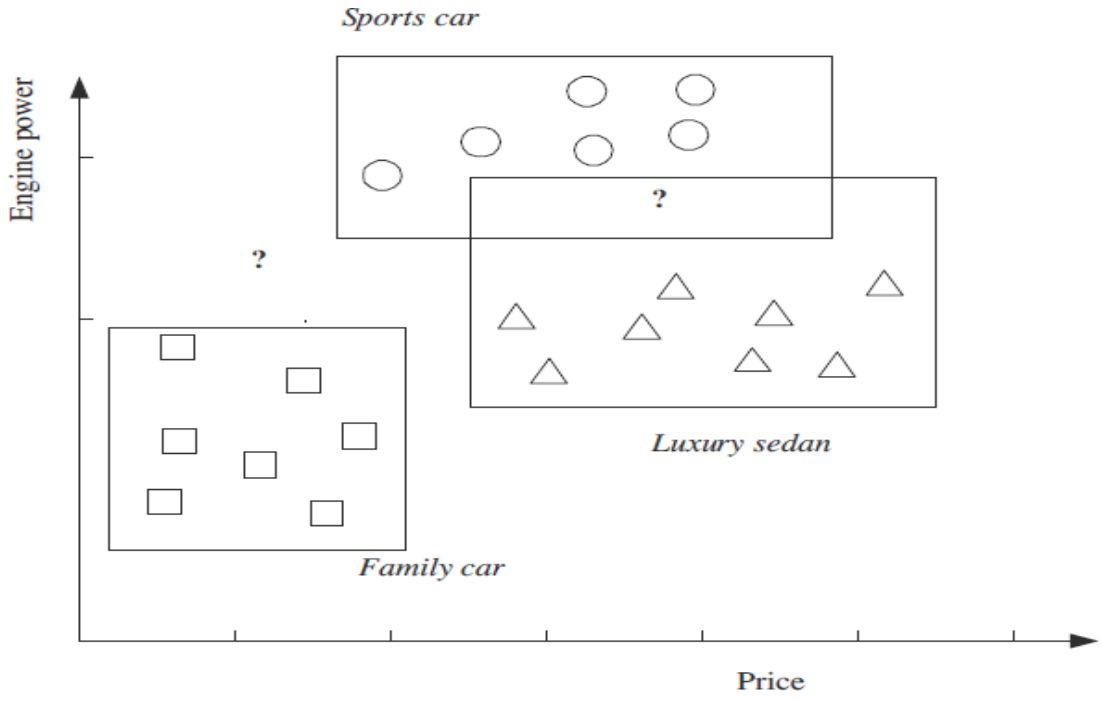
In our example of learning a family car, we have positive examples belonging to the class family car and the negative examples belonging to all other cars. This is a *two-class* problem. In the general case, we have *K* classes denoted as C*i*, *i* = 1, . . . , *K*, and an input instance belongs to one and exactly one of them. The training set is now of the form,

$$\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{r}^t\}_{t=1}^N$$

$$r_i^t = \begin{cases} 1 \text{ if } \boldsymbol{x}^t \in C_i \\ 0 \text{ if } \boldsymbol{x}^t \in C_j,\ j \neq i \end{cases}$$

Train hypotheses
$h_i(\boldsymbol{x}),\ i = 1,\ldots,K$:

$$h_i(\boldsymbol{x}^t) = \begin{cases} 1 \text{ if } \boldsymbol{x}^t \in C_i \\ 0 \text{ if } \boldsymbol{x}^t \in C_j,\ j \neq i \end{cases}$$



There are three classes: family car, sports car, and luxury sedan. There are three hypotheses induced, each one covering the instances of one class and leaving outside the instances of the other two classes .'?' are reject regions where no, or more than one, class is chosen.

Model Selection & Generalization:

- Learning is an ill-posed problem; data is not sufficient to find a unique solution
- The need for inductive bias, assumptions about H
- Generalization: How well a model performs on new data
- Overfitting: H more complex than C or f
- Underfitting: H less complex than C or f

Triple Trade-Off:

- There is a trade-off between three factors (Dietterich, 2003):
    1. Complexity of H, *c* (H),
    2. Training set size, *N*,
    3. Generalization error, *E*, on new data

        As *N* increases, E decreasesAs *c* (H), first *E* decreases and then *E* increases

Cross-Validation:

- To estimate generalization error, we need data unseen during training. We split the data as
    - Training set (50%)
    - Validation set (25%)
    - Test (publication) set (25%)
- Resampling when there is few data

Dimensions of a Supervised Learner:

- Model : $g(\boldsymbol{x} \mid \theta)$
- Loss function: $E(\theta \mid \mathcal{X}) = \sum_t L(r^t, g(\boldsymbol{x}^t \mid \theta))$
- Optimization procedure:

$$\theta^* = \arg\min_\theta E(\theta \mid \mathcal{X})$$

Applications of Machine learning

Machine learning is a buzzword for today's technology, and it is growing very rapidly day by day. We are using machine learning in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc. Below are some most trending real-world applications of Machine Learning:

1. Image Recognition:

Image recognition is one of the most common applications of machine learning. It is used to identify objects, persons, places, digital images, etc. The popular use case of image recognition and face detection is, **Automatic friend tagging suggestion**:

Facebook provides us a feature of auto friend tagging suggestion. Whenever we upload a photo with our Facebook friends, then we automatically get a tagging suggestion with name, and the technology behind this is machine learning's **face detection** and **recognition algorithm**.

It is based on the Facebook project named "**Deep Face**," which is responsible for face recognition and person identification in the picture.

2. Speech Recognition

While using Google, we get an option of "**Search by voice**," it comes under speech recognition, and it's a popular application of machine learning.

Speech recognition is a process of converting voice instructions into text, and it is also known as "**Speech to text**", or "**Computer speech recognition**." At present, machine learning algorithms are widely used by various applications of speech recognition. **Google assistant**, **Siri**, **Cortana**, and **Alexa** are using speech recognition technology to follow the voice instructions.

3. Traffic prediction:

If we want to visit a new place, we take help of Google Maps, which shows us the correct path with the shortest route and predicts the traffic conditions.

It predicts the traffic conditions such as whether traffic is cleared, slow-moving, or heavily congested with the help of two ways:

- **Real Time location** of the vehicle form Google Map app and sensors
- **Average time has taken** on past days at the same time.

Everyone who is using Google Map is helping this app to make it better. It takes information from the user and sends back to its database to improve the performance.

4. Product recommendations:

Machine learning is widely used by various e-commerce and entertainment companies such as **Amazon**, **Netflix**, etc., for product recommendation to the user. Whenever we search for some product on Amazon, then we started getting an advertisement for the same product while internet surfing on the same browser and this is because of machine learning.

Google understands the user interest using various machine learning algorithms and suggests the product as per customer interest.

As similar, when we use Netflix, we find some recommendations for entertainment series, movies, etc., and this is also done with the help of machine learning.

5. Self-driving cars:

One of the most exciting applications of machine learning is self-driving cars. Machine learning plays a significant role in self-driving cars. Tesla, the most popular car manufacturing company is working on self-driving car. It is using unsupervised learning method to train the car models to detect people and objects while driving.

6. Email Spam and Malware Filtering:

Whenever we receive a new email, it is filtered automatically as important, normal, and spam. We always receive an important mail in our inbox with the important symbol and spam emails in our spam box, and the technology behind this is Machine learning. Below are some spam filters used by Gmail:

- Content Filter
- Header filter
- General blacklists filter
- Rules-based filters
- Permission filters

Some machine learning algorithms such as **Multi-Layer Perceptron**, **Decision tree**, and **Naïve Bayes classifier** are used for email spam filtering and malware detection.

7. Virtual Personal Assistant:

We have various virtual personal assistants such as **Google assistant**, **Alexa**, **Cortana**, **Siri**. As the name suggests, they help us in finding the information using our voice instruction. These assistants can help us in various ways just by our voice instructions such as Play music, call someone, Open an email, Scheduling an appointment, etc.

These virtual assistants use machine learning algorithms as an important part.

These assistant record our voice instructions, send it over the server on a cloud, and decode it using ML algorithms and act accordingly.

8. Online Fraud Detection:

Machine learning is making our online transaction safe and secure by detecting fraud transaction. Whenever we perform some online transaction, there may be various ways that a fraudulent transaction can take place such as **fake accounts**, **fake ids**, and **steal money** in the middle of a transaction. So to detect this, **Feed Forward Neural network** helps us by checking whether it is a genuine transaction or a fraud transaction.

For each genuine transaction, the output is converted into some hash values, and these values become the input for the next round. For each genuine transaction, there is a specific pattern which gets change for the fraud transaction hence, it detects it and makes our online transactions more secure.

9. Stock Market trading:

Machine learning is widely used in stock market trading. In the stock market, there is always a risk of up and downs in shares, so for this machine learning's **long short term memory neural network** is used for the prediction of stock market trends.

10. Medical Diagnosis:

In medical science, machine learning is used for diseases diagnoses. With this, medical technology is growing very fast and able to build 3D models that can predict the exact position of lesions in the brain.

It helps in finding brain tumors and other brain-related diseases easily.

11. Automatic Language Translation:

Nowadays, if we visit a new place and we are not aware of the language then it is not a problem at all, as for this also machine learning helps us by converting the text into our known languages. Google's GNMT (Google Neural Machine Translation) provide this feature, which is a Neural Machine Learning that translates the text into our familiar language, and it called as automatic translation.

The technology behind the automatic translation is a sequence to sequence learning algorithm, which is used with image recognition and translates the text from one language to another language.

Supervised learning Vs Unsupervised Learning

Supervised Learning	Unsupervised Learning
Supervised learning algorithms are trained using labeled data.	Unsupervised learning algorithms are trained using unlabeled data.
Supervised learning model takes direct feedback to check if it is predicting correct output or not.	Unsupervised learning model does not take any feedback.
Supervised learning model predicts the output.	Unsupervised learning model finds the hidden patterns in data.
In supervised learning, input data is provided to the model along with the output.	In unsupervised learning, only input data is provided to the model.
The goal of supervised learning is to train the model so that it can predict the output when it is given new data.	The goal of unsupervised learning is to find the hidden patterns and useful insights from the unknown dataset.
Supervised learning needs supervision to train the model.	Unsupervised learning does not need any supervision to train the model.
Supervised learning can be categorized in **Classification** and **Regression** problems.	Unsupervised Learning can be classified in **Clustering** and **Associations** problems.

Supervised learning can be used for those cases where we know the input as well as corresponding outputs.	Unsupervised learning can be used for those cases where we have only input data and no corresponding output data.
Supervised learning model produces an accurate result.	Unsupervised learning model may give less accurate result as compared to supervised learning.
Supervised learning is not close to true Artificial intelligence as in this, we first train the model for each data, and then only it can predict the correct output.	Unsupervised learning is more close to the true Artificial Intelligence as it learns similarly as a child learns daily routine things by his experiences.
It includes various algorithms such as Linear Regression, Logistic Regression, Support Vector Machine, Multi-class Classification, Decision tree, Bayesian Logic, etc.	It includes various algorithms such as Clustering, KNN, and Apriori algorithm.

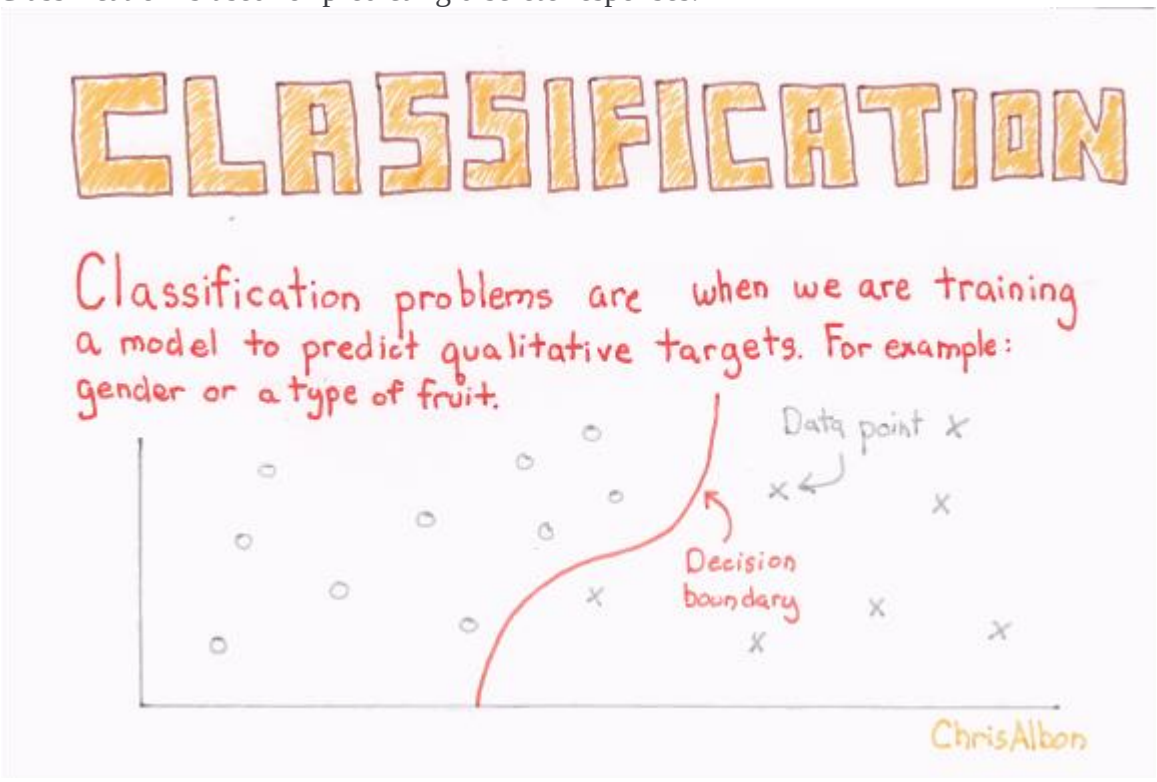
Supervised Learning

In supervised learning, algorithms learn from labeled data. After understanding the data, the algorithm determines which label should be given to new data by associating patterns to the unlabeled new data.
Supervised learning can be divided into two categories: classification and regression.

Classification

Classification is a technique for determining which class the dependent belongs to based on one or more independent variables.

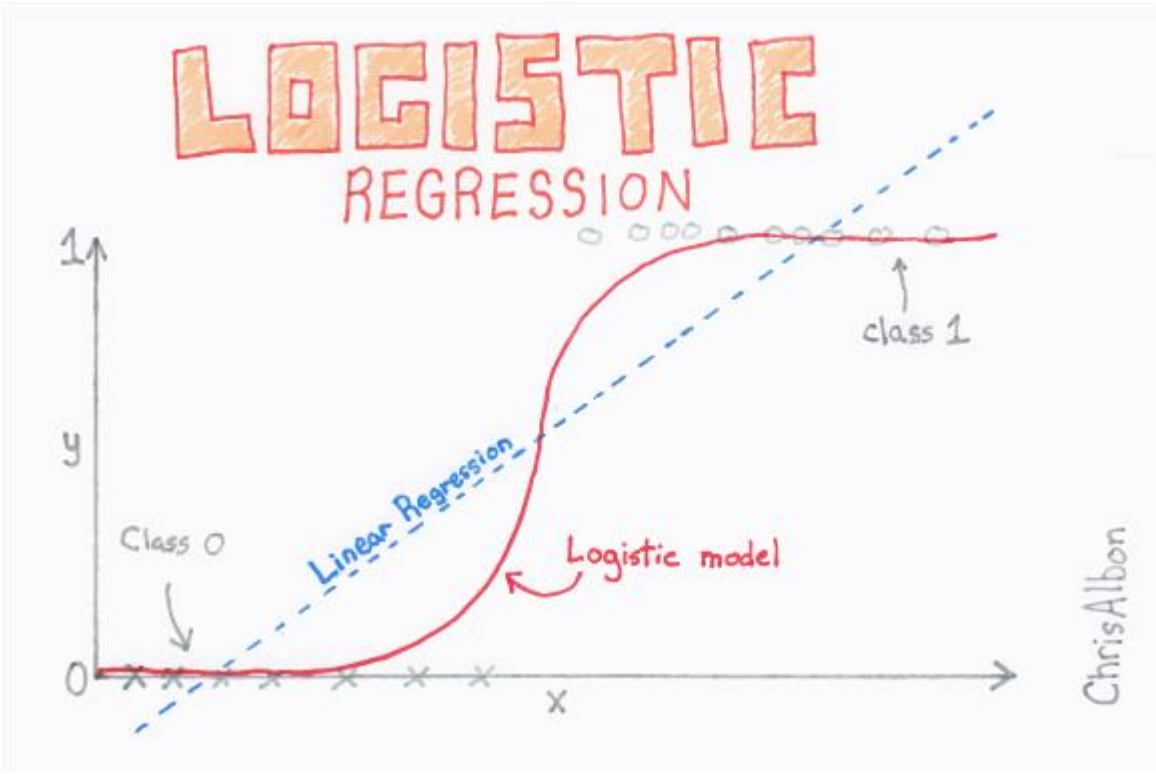
Classification is used for predicting discrete responses.



1. LOGISTIC REGRESSION

Logistic regression is kind of like linear regression, but is used when the dependent variable is not a number but something else (e.g., a "yes/no" response). It's called regression but performs

classification based on the regression and it classifies the dependent variable into either of the classes.



Logistic regression is used for prediction of output which is binary, as stated above. For example, if a credit card company builds a model to decide whether or not to issue a credit card to a customer, it will model for whether the customer is going to “default” or “not default” on their card.

$$y = b_0 + b_1 x$$

Linear Regression

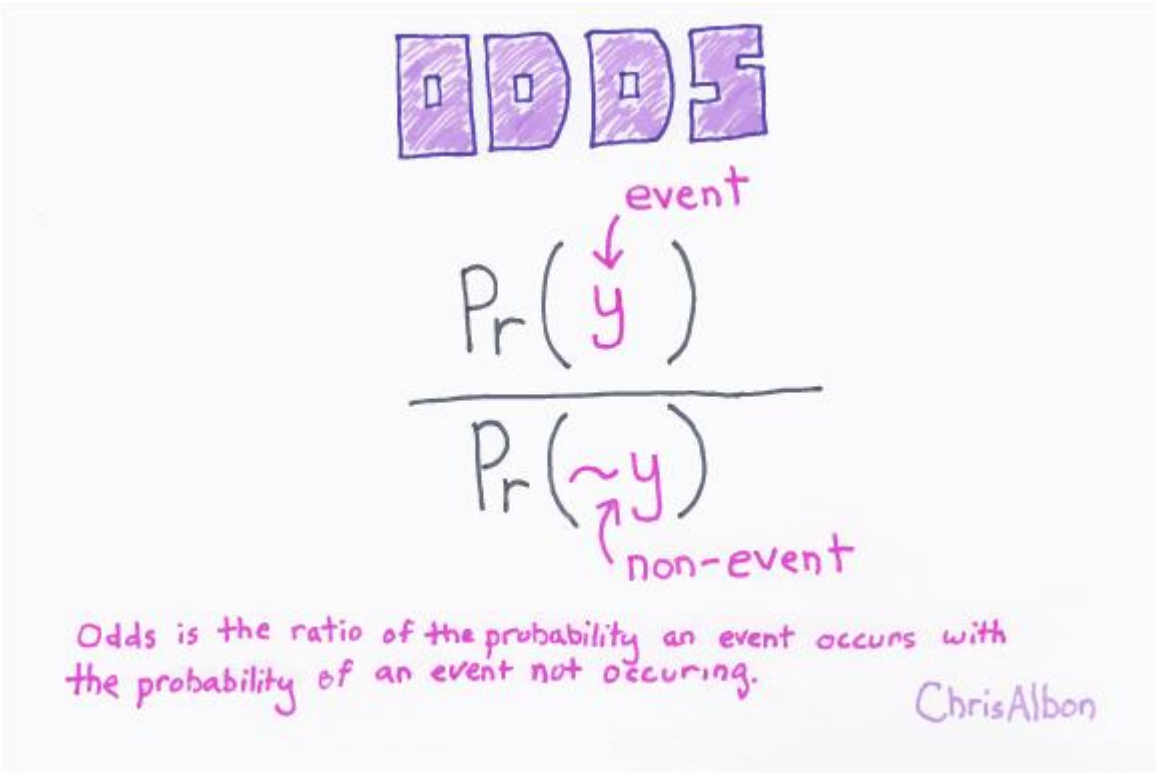
Firstly, linear regression is performed on the relationship between variables to get the model. The threshold for the classification line is assumed to be at 0.5.

$$p = \frac{1}{1 + e^{-y}}$$

Logistic Sigmoid Function

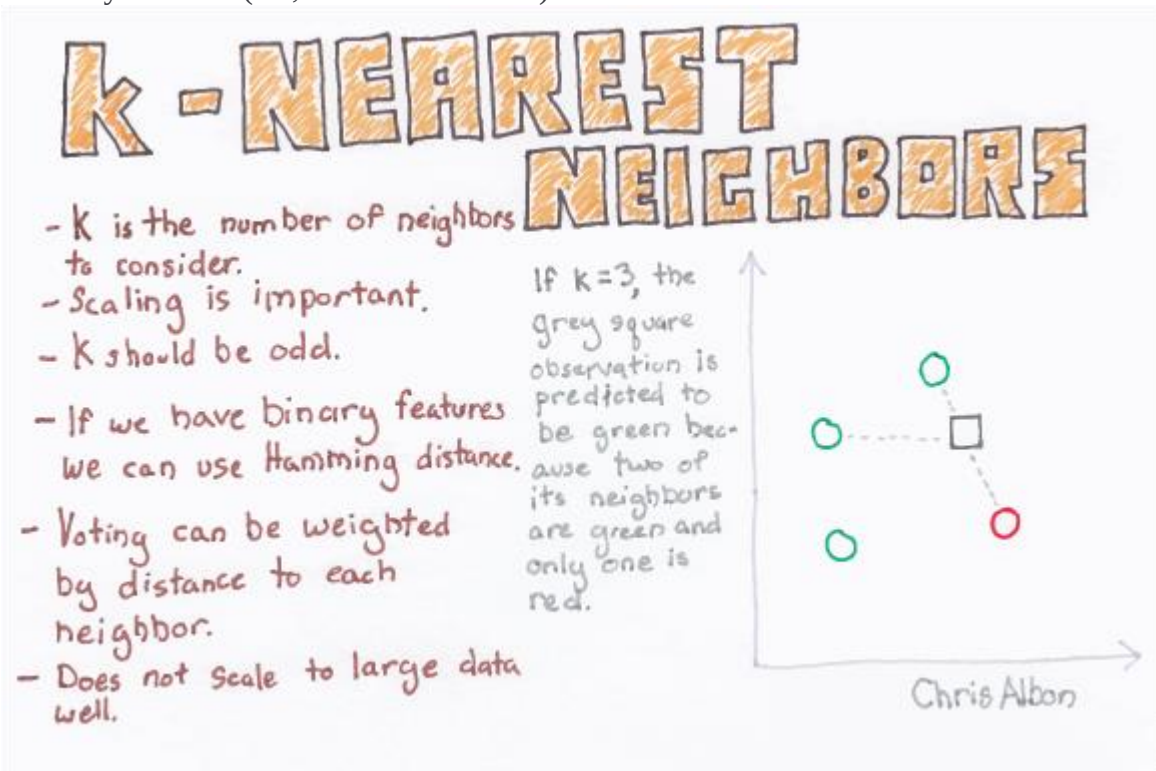
Logistic function is applied to the regression to get the probabilities of it belonging in either class.

It gives the log of the probability of the event occurring to the log of the probability of it not occurring. In the end, it classifies the variable based on the higher probability of either class.



2. K-NEAREST NEIGHBORS (K-NN)

K-NN algorithm is one of the simplest classification algorithms and it is used to identify the data points that are separated into several classes to predict the classification of a new sample point. K-NN is a non-parametric**,** lazy learning algorithm. It classifies new cases based on a similarity measure (i.e., distance functions).



DOES K-NN LEARN

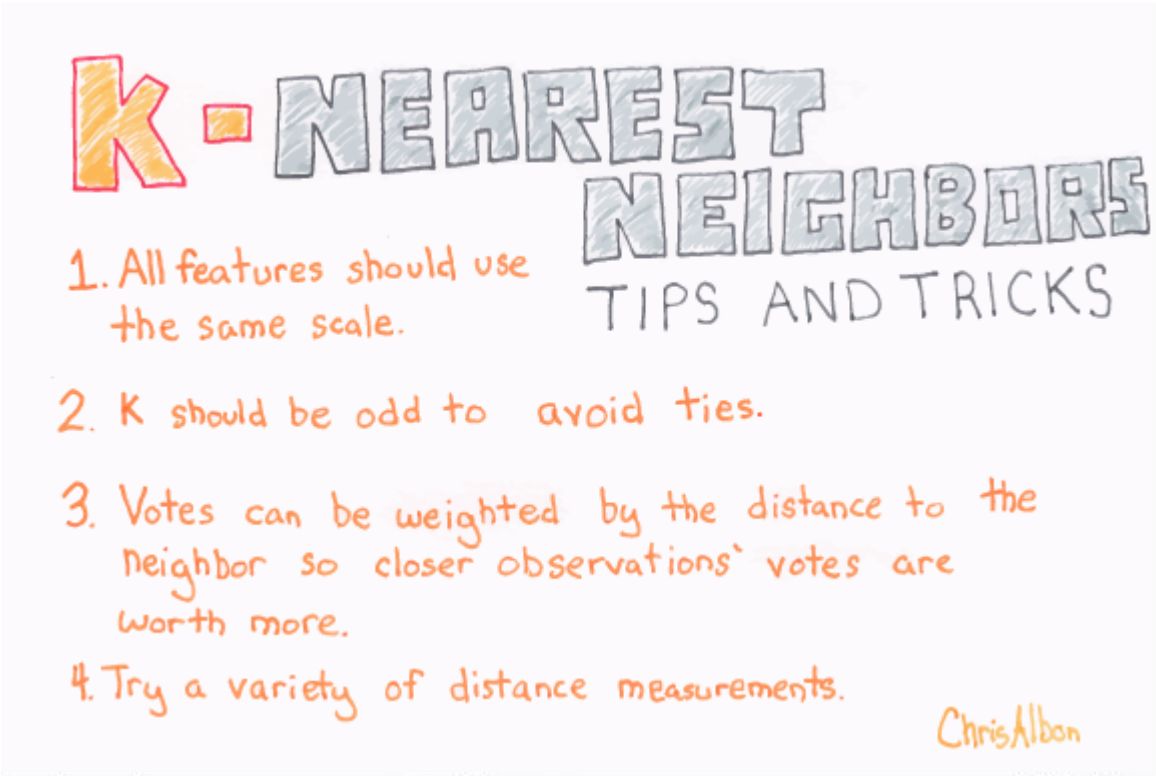
K-nearest neighor does not "learn" per-se. It is lazy and just memorizes the data.

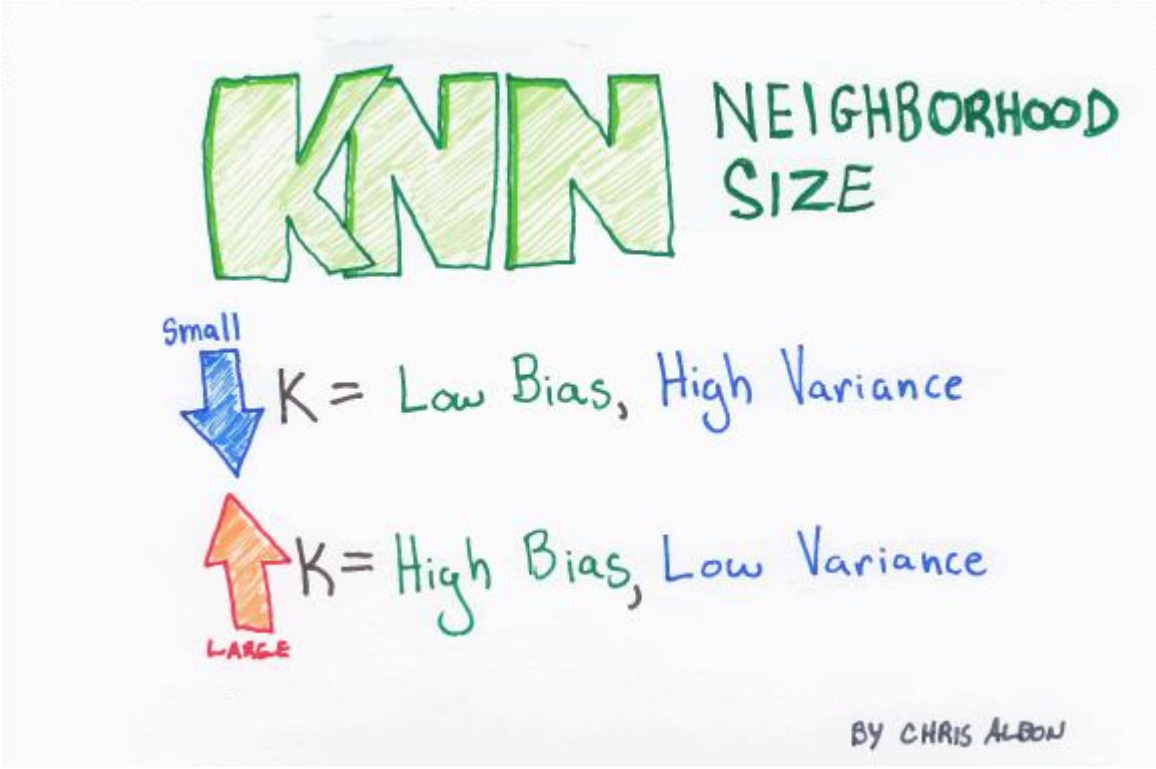
Chris Albon

DOES K-NN LEARN

K-nearest neighor does not "learn" per-se. It is lazy and just memorizes the data.

Chris Albon

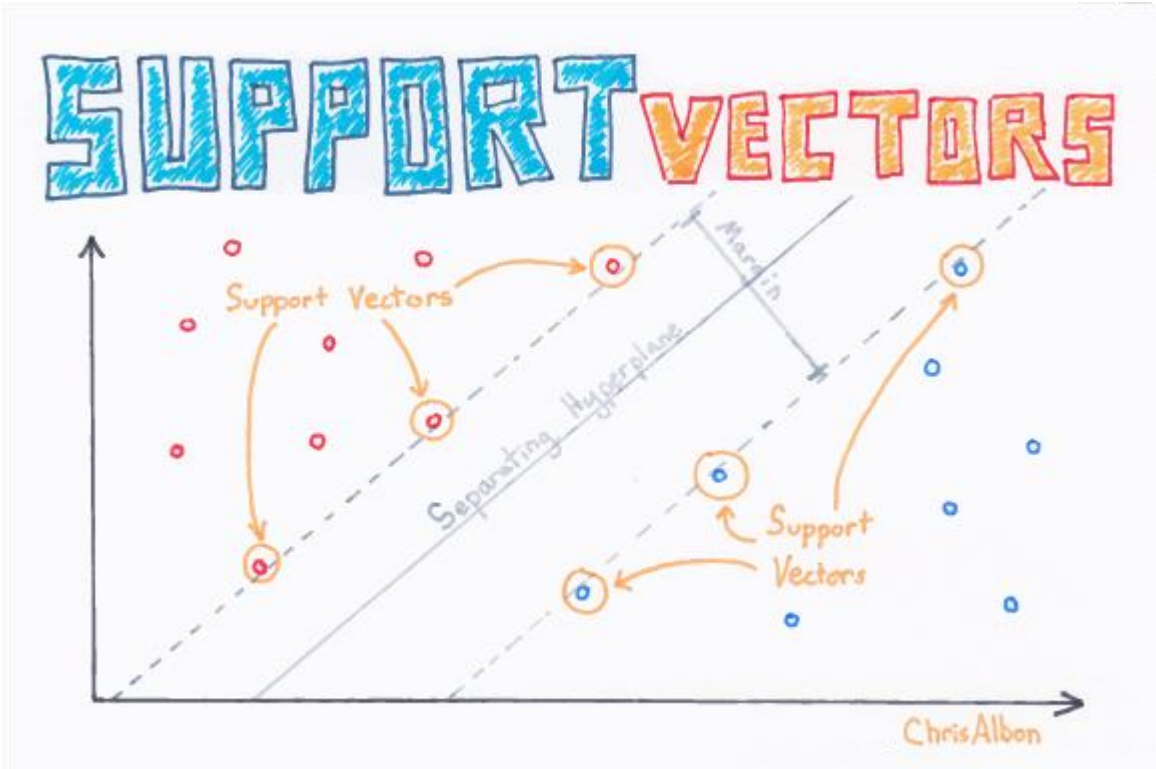




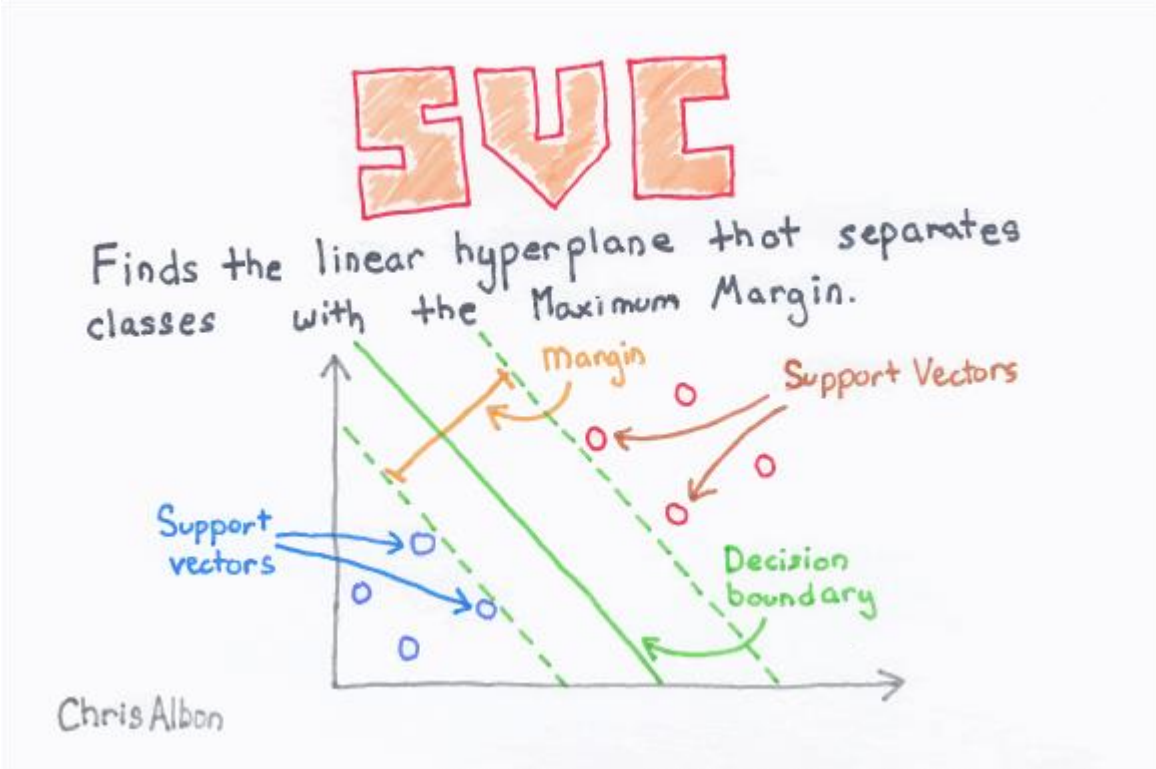
K-NN works well with a small number of input variables (*p*), but struggles when the number of inputs is very large.

3. SUPPORT VECTOR MACHINE (SVM)

Support vector is used for both regression and classification. It is based on the concept of decision planes that define decision boundaries. A decision plane (hyperplane) is one that separates between a set of objects having different class memberships.



It performs classification by finding the hyperplane that maximizes the margin between the two classes with the help of support vectors.



The learning of the hyperplane in SVM is done by transforming the problem using some linear algebra (i.e., the example above is a linear kernel which has a linear separability between each variable).

For higher dimensional data, other kernels are used as points and cannot be classified easily. They are specified in the next section.

Kernel SVM

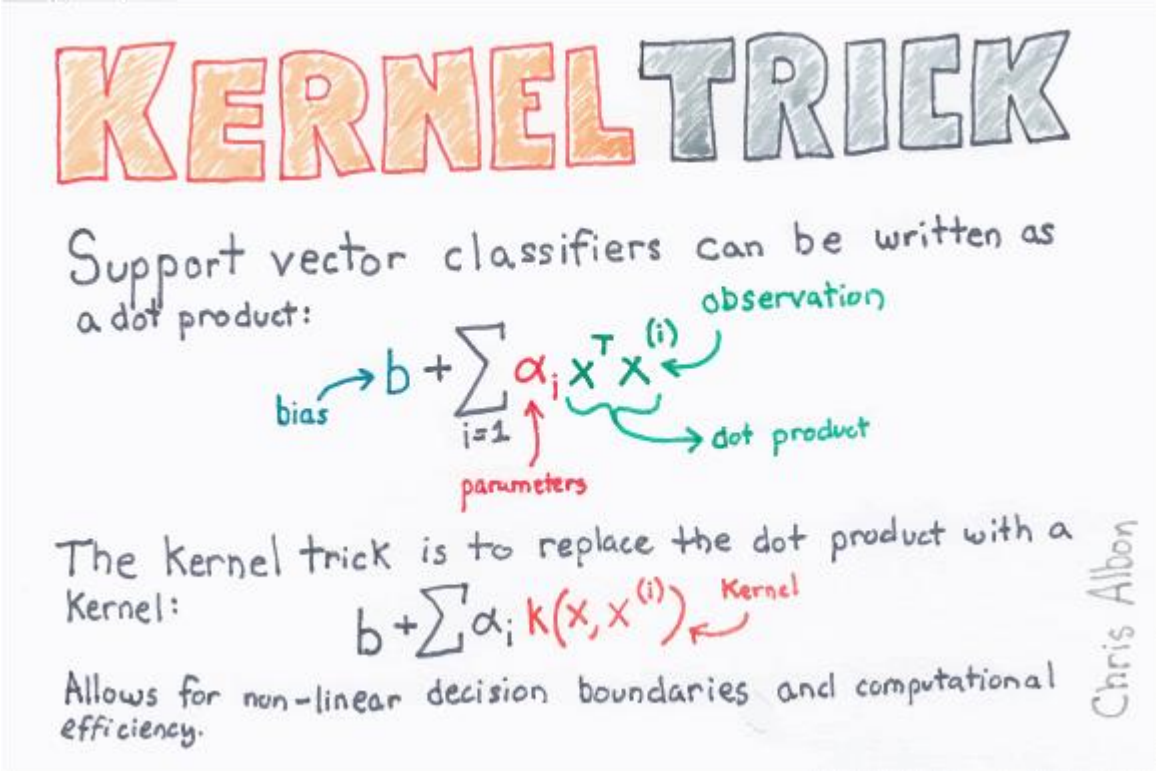
Kernel SVM takes in a kernel function in the SVM algorithm and transforms it into the required form that maps data on a higher dimension which is separable.

Types of kernel function are:

$$K(\mathbf{X_i}, \mathbf{X_j}) = \begin{Bmatrix} \mathbf{X_i} \cdot \mathbf{X_j} & \text{Linear} \\ (\gamma \mathbf{X_i} \cdot \mathbf{X_j} + C)^d & \text{Polynomial} \\ \exp(-\gamma |\mathbf{X_i} - \mathbf{X_j}|^2) & \text{RBF} \\ \tanh(\gamma \mathbf{X_i} \cdot \mathbf{X_j} + C) & \text{Sigmoid} \end{Bmatrix}$$

Type of kernel functions

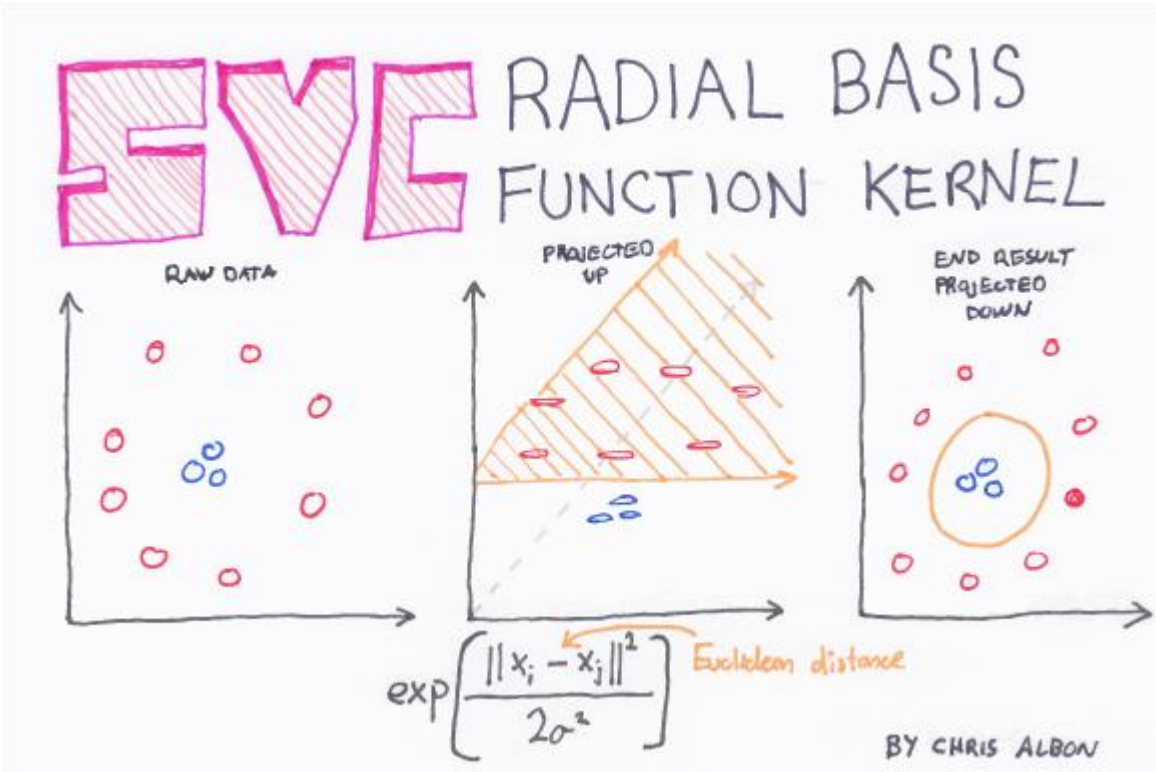
1. Linear SVM is the one we discussed earlier.
2. In polynomial kernel, the degree of the polynomial should be specified. It allows for curved lines in the input space.
3. In the radial basis function (RBF) kernel, it is used for non-linearly separable variables. For distance, metric squared Euclidean distance is used. Using a typical value of the parameter can lead to overfitting our data. It is used by default in sklearn.
4. Sigmoid kernel, similar to logistic regression is used for binary classification.



Kernel trick uses the kernel function to transform data into a higher dimensional feature space and makes it possible to perform the linear separation for classification.

Radial Basis Function (RBF) Kernel

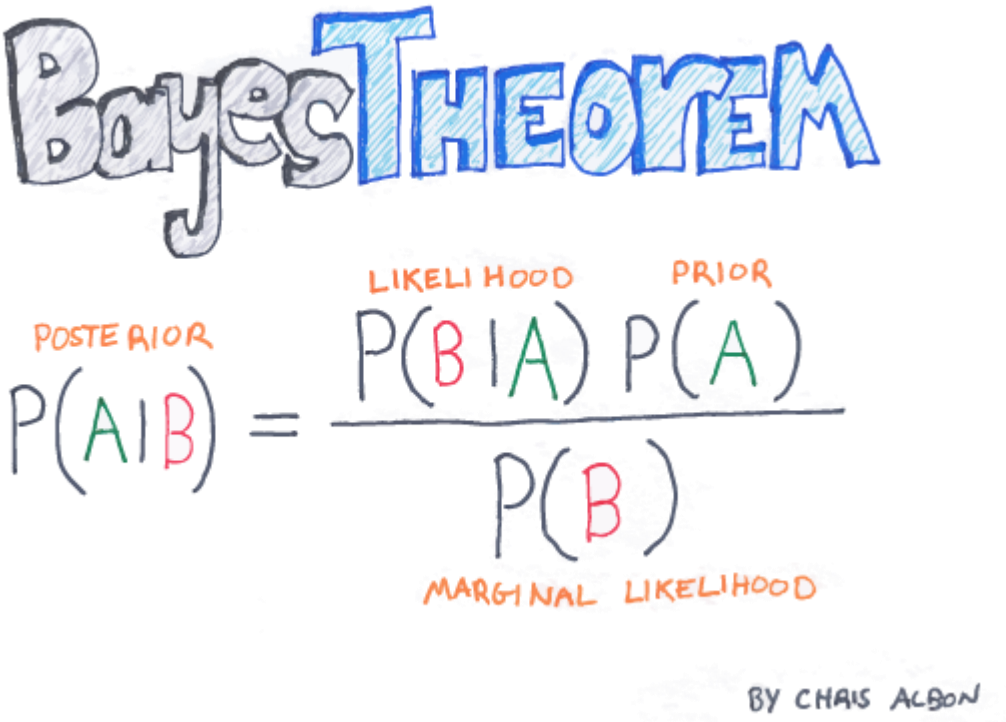
The RBF kernel SVM decision region is actually also a linear decision region. What RBF kernel SVM actually does is create non-linear combinations of features to uplift the samples onto a higher-dimensional feature space where a linear decision boundary can be used to separate classes.



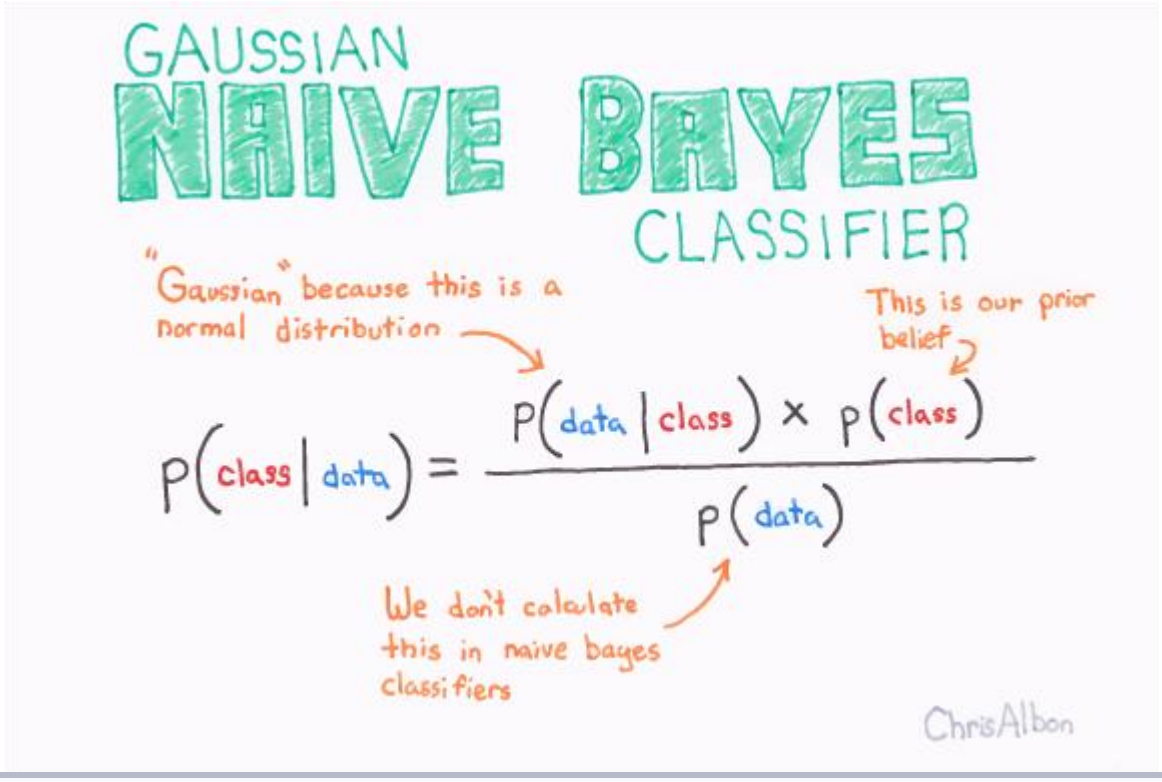
So, the rule of thumb is: use linear SVMs for linear problems, and nonlinear kernels such as the RBF kernel for non-linear problems.

4. NAIVE BAYES

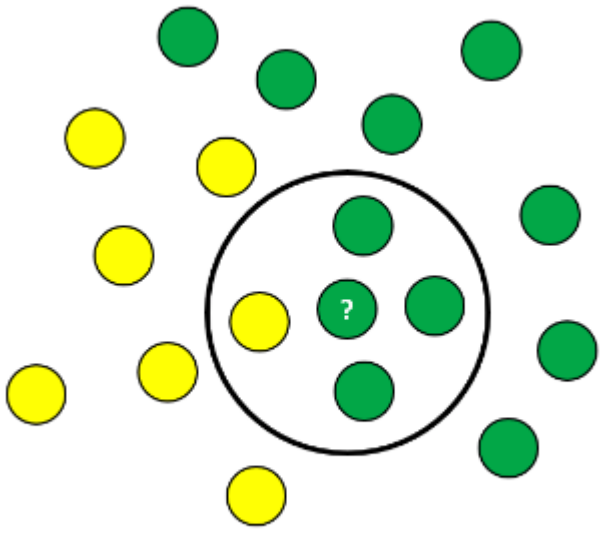
The naive Bayes classifier is based on Bayes' theorem with the independence assumptions between predictors (i.e., it assumes the presence of a feature in a class is unrelated to any other feature). Even if these features depend on each other, or upon the existence of the other features, all of these properties independently. Thus, the name naive Bayes.



Based on naive Bayes, Gaussian naive Bayes is used for classification based on the binomial (normal) distribution of data.



- P(class|data) is the posterior probability of class(target) given predictor(attribute). The probability of a data point having either class, given the data point. This is the value that we are looking to calculate.
- P(class) is the prior probability of class.
- P(data|class) is the likelihood, which is the probability of predictor given class.
- P(data) is the prior probability of predictor or marginal likelihood.



NB Classification Example

Steps

1. Calculate Prior Probability

P(class) = Number of data points in the class/Total no. of observations

P(yellow) = 10/17

P(green) = 7/17

2. Calculate Marginal Likelihood

P(data) = Number of data points similar to observation/Total no. of observations

P(?) = 4/17

The value is present in checking both the probabilities.

3. Calculate Likelihood

P(data/class) = Number of similar observations to the class/Total no. of points in the class.

P(?/yellow) = 1/7

P(?/green) = 3/10

4. Posterior Probability for Each Class

$$p(class/data) = \frac{P(data/class) * P(class)}{P(data)}$$

$$P(yellow/?) = \frac{1/7 * 7/17}{4/17} = 0.25$$

$$P(green/?) = \frac{3/10 * 10/17}{4/17} = 0.75$$

5. Classification

$$P(class1/data) > P(class2/data)$$

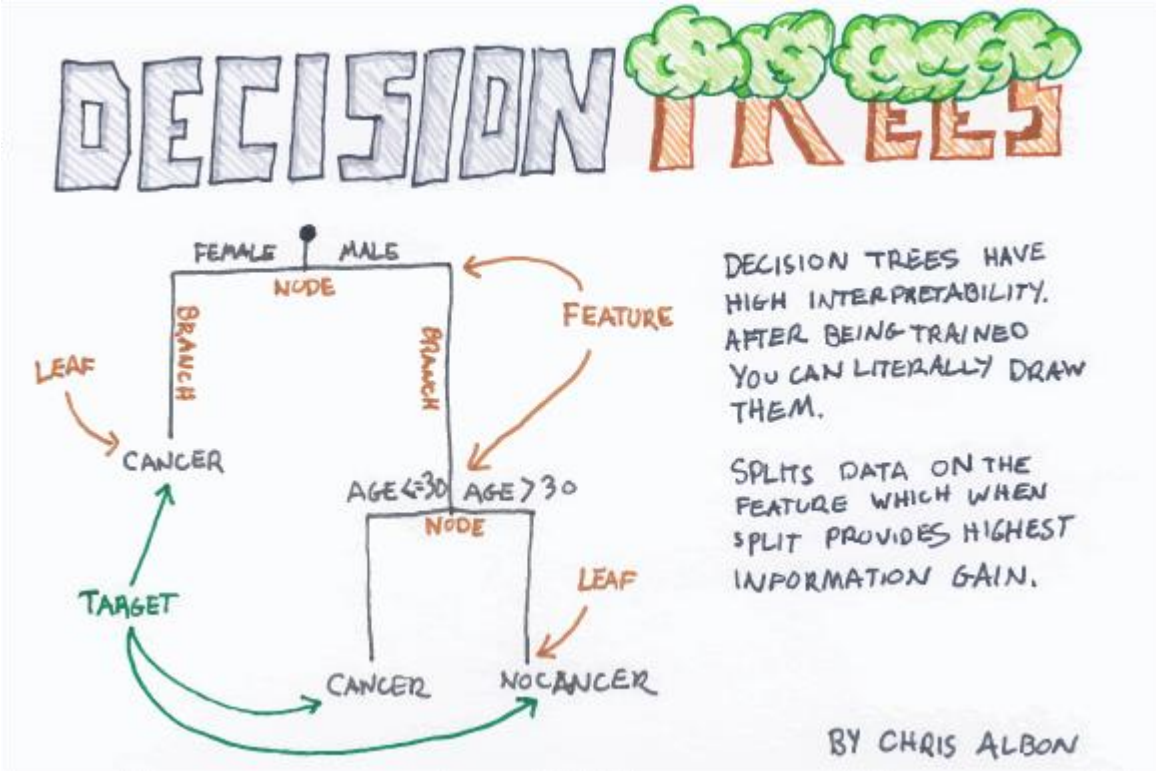
$$P(green/?) > P(yellow/?)$$

The higher probability, the class belongs to that category as from above 75% probability the point belongs to class green.

Multinomial, Bernoulli naive Bayes are the other models used in calculating probabilities. Thus, a naive Bayes model is easy to build, with no complicated iterative parameter estimation, which makes it particularly useful for very large datasets.

5. DECISION TREE CLASSIFICATION

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. It follows Iterative Dichotomiser 3(ID3) algorithm structure for determining the split.



Entropy and information gain are used to construct a decision tree.

Entropy

Entropy is the degree or amount of uncertainty in the randomness of elements. In other words, it is a measure of impurity**.**

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

Entropy

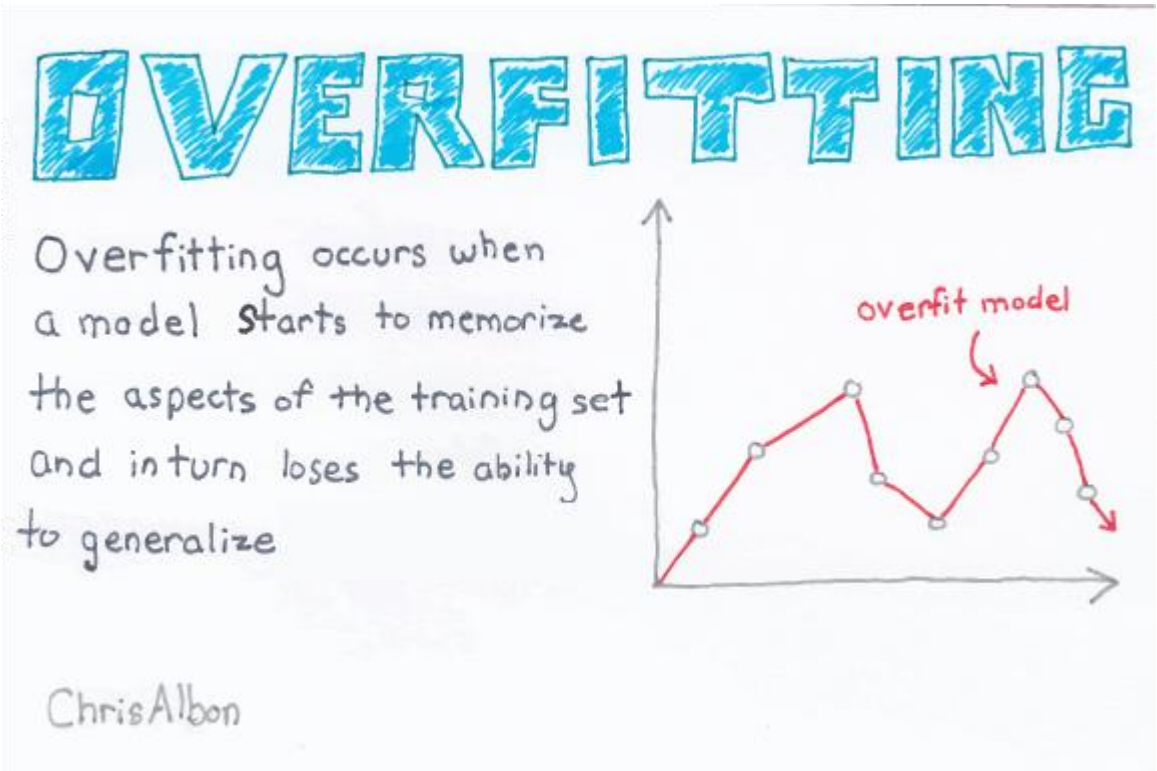
Intuitively, it tells us about the predictability of a certain event. Entropy calculates the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero, and if the sample is equally divided it has an entropy of one.

Information Gain
Information gain measures the relative change in entropy with respect to the independent attribute. It tries to estimate the information contained by each attribute. Constructing a decision tree is all about finding the attribute that returns the highest information gain (i.e., the most homogeneous branches).

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

Where *Gain(T, X)* is the information gain by applying feature *X*. *Entropy(T)* is the entropy of the entire set, while the second term calculates the entropy after applying the feature *X*. Information gain ranks attributes for filtering at a given node in the tree. The ranking is based on the highest information gain entropy in each split.

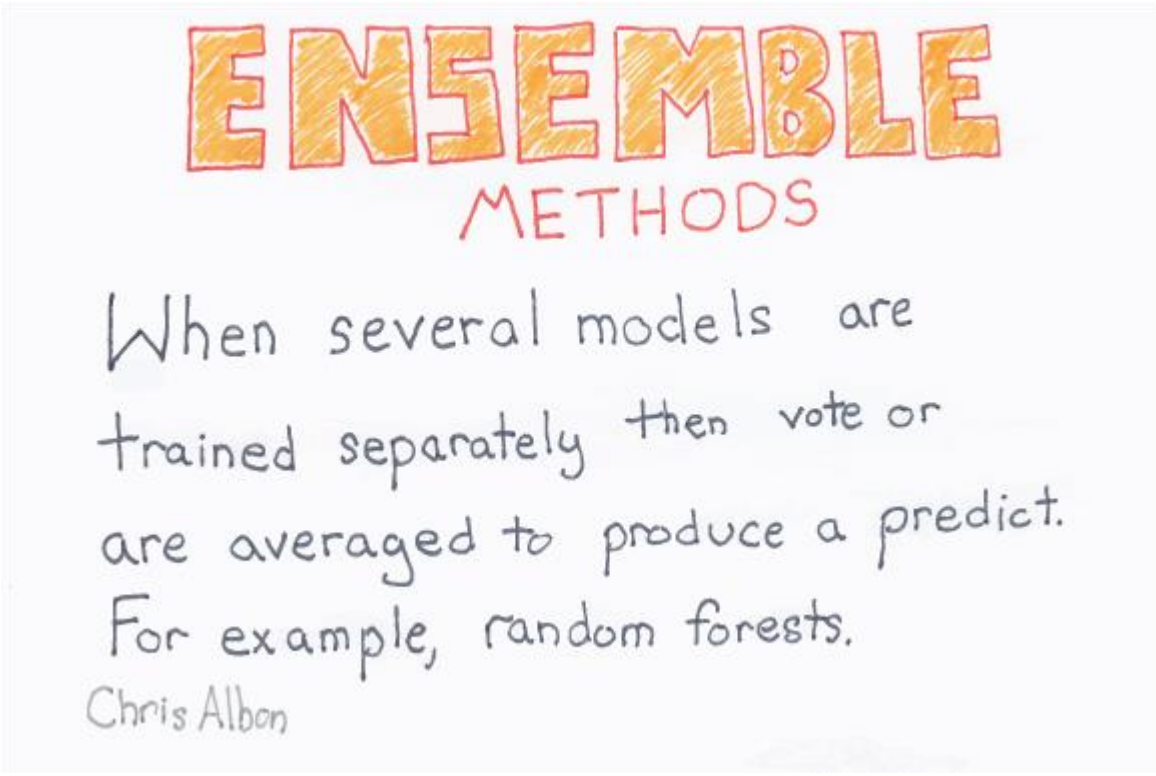
The disadvantage of a decision tree model is overfitting, as it tries to fit the model by going deeper in the training set and thereby reducing test accuracy.



Overfitting in decision trees can be minimized by pruning nodes.

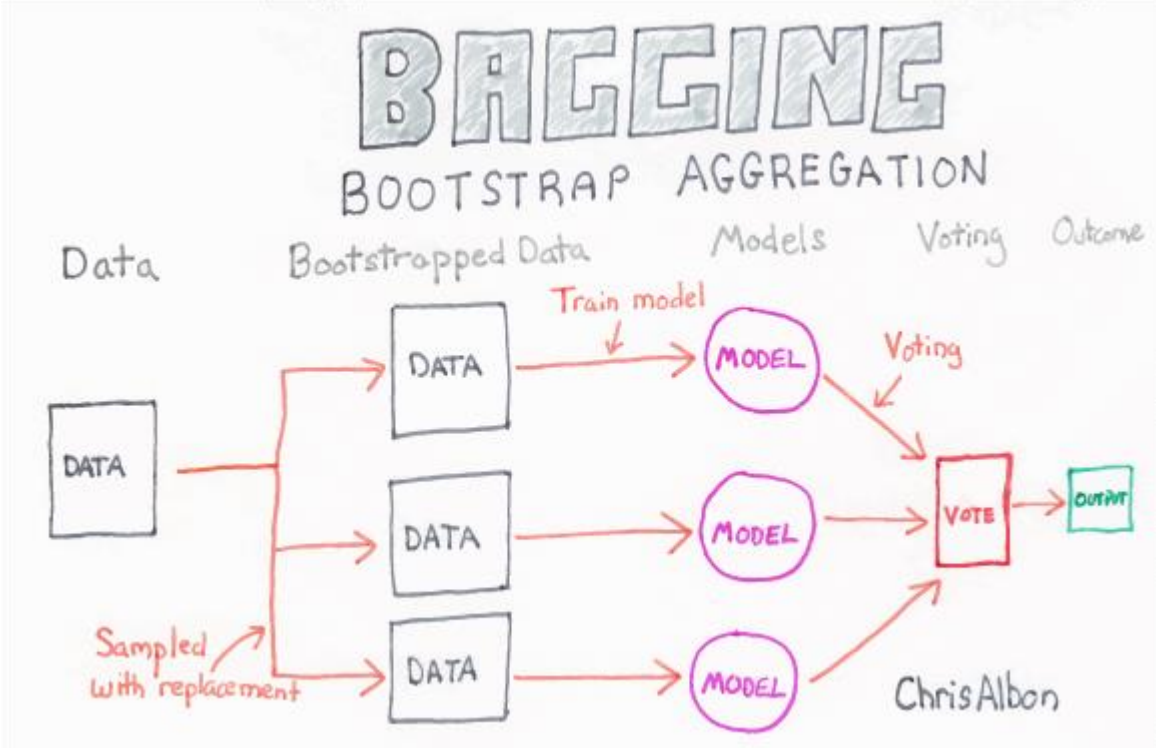
Ensemble Methods for Classification

An ensemble model is a *team of models*. Technically, ensemble models comprise several supervised learning models that are individually trained and the results merged in various ways to achieve the final prediction. This result has higher predictive power than the results of any of its constituting learning algorithms independently.

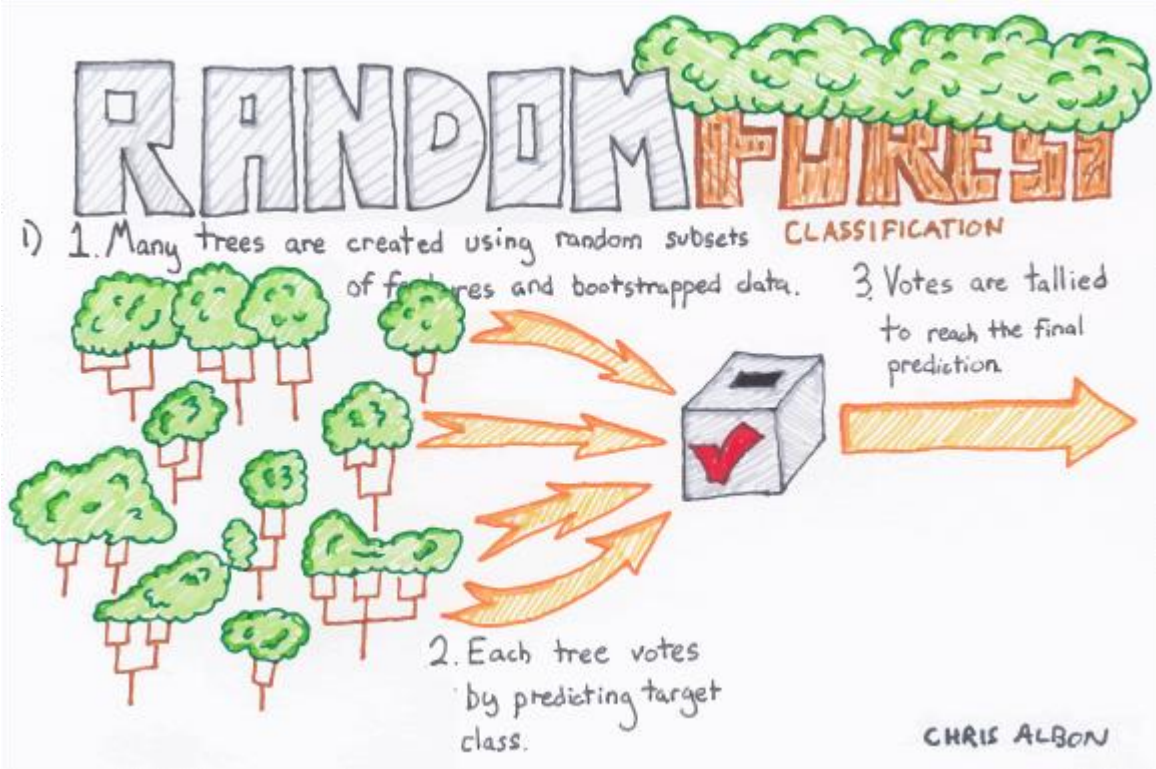


1. RANDOM FOREST CLASSIFICATION

Random forest classifier is an ensemble algorithm based on bagging i.e bootstrap aggregation. Ensemble methods combines more than one algorithm of the same or different kind for classifying objects (i.e., an ensemble of SVM, naive Bayes or decision trees, for example.)



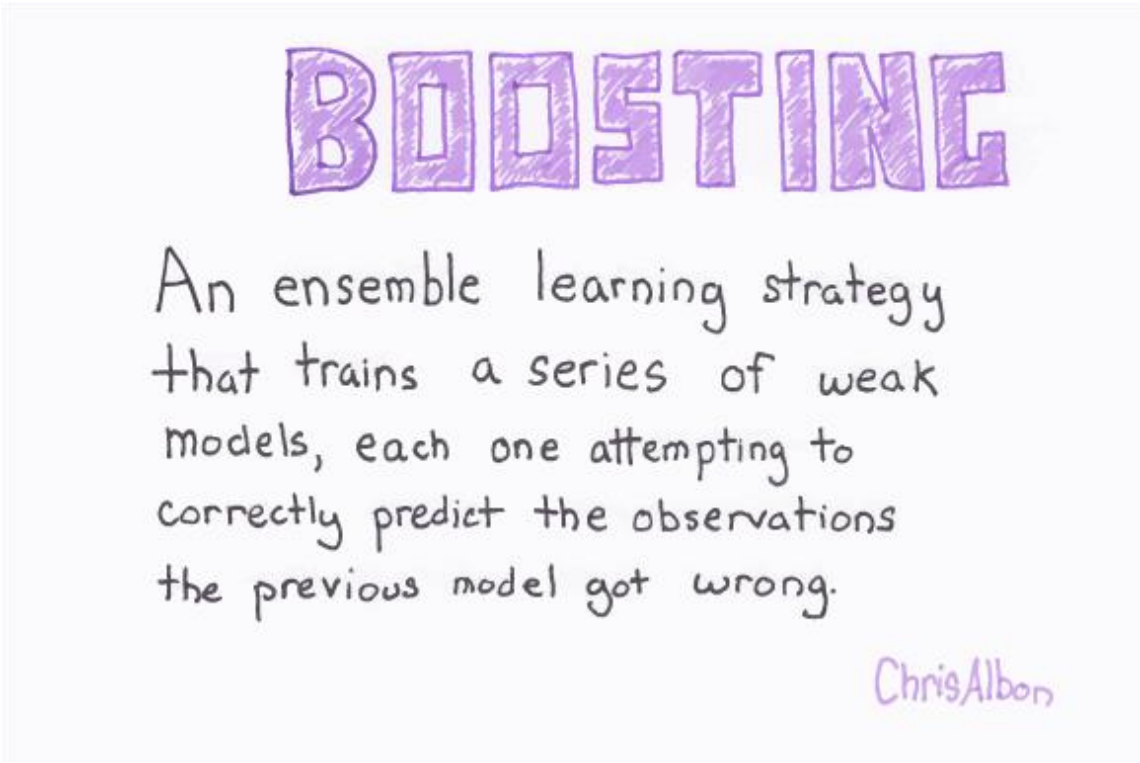
The general idea is that a combination of learning models increases the overall result selected.



Deep decision trees may suffer from overfitting, but random forests prevent overfitting by creating trees on random subsets. The main reason is that it takes the average of all the predictions, which cancels out the biases.Random forest adds additional randomness to the model while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

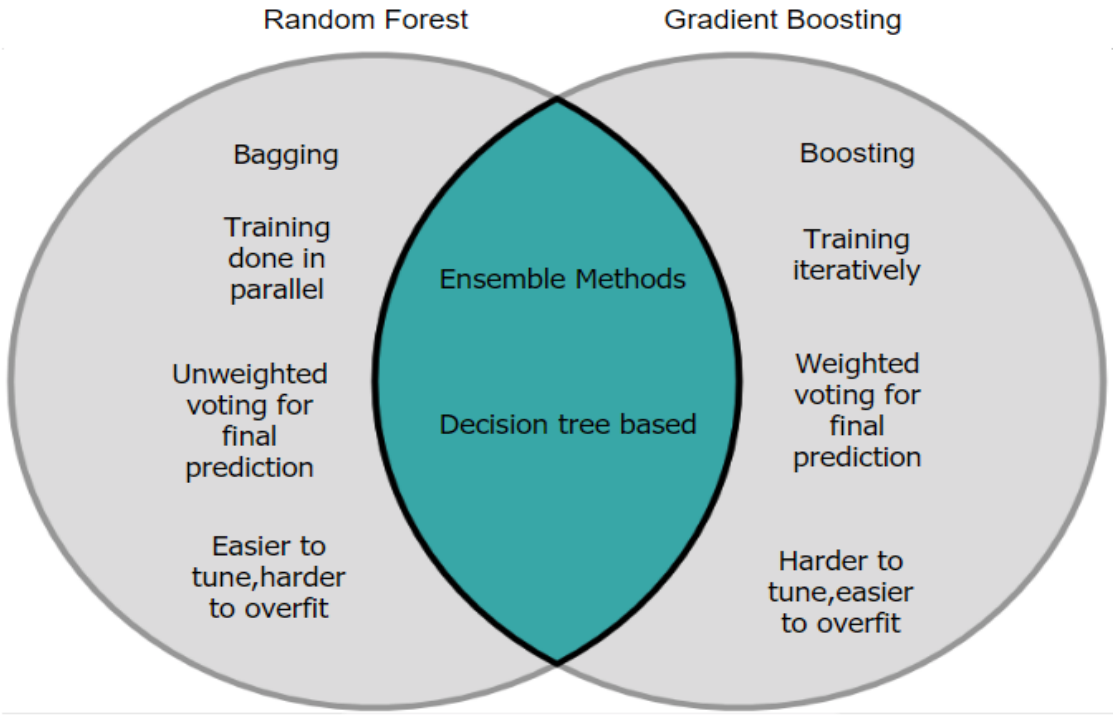
2. GRADIENT BOOSTING CLASSIFICATION

Gradient boosting classifier is a boosting ensemble method. Boosting is a way to combine (ensemble) weak learners, primarily to reduce prediction bias. Instead of creating a pool of predictors, as in bagging, boosting produces a cascade of them, where each output is the input for the following learner. Typically, in a bagging algorithm trees are grown in parallel to get the average prediction across all trees, where each tree is built on a sample of original data. Gradient boosting, on the other hand, takes a sequential approach to obtaining predictions instead of parallelizing the tree building process. In gradient boosting, each decision tree predicts the error of the previous decision tree—thereby *boosting* (improving) the error (gradient).



Working of Gradient Boosting

1. Initialize predictions with a simple decision tree.
2. Calculate residual (actual-prediction) value.
3. Build another shallow decision tree that predicts residual based on all the independent values.
4. Update the original prediction with the new prediction multiplied by learning rate.
5. Repeat steps two through four for a certain number of iterations (the number of iterations will be the number of trees).



Difference between RF & GB

Unsupervised classification is where the outcomes (groupings of pixels with common characteristics) are based on the software analysis of an image without the user providing sample classes. The computer uses techniques to determine which pixels are related and groups them into classes. The user can specify which algorism the software will use and the desired number of output classes but otherwise does not aid in the classification process. However, the user must have knowledge of the area being classified when the groupings of pixels with common characteristics produced by the computer have to be related to actual features on the ground (such as wetlands, developed areas, coniferous forests, etc.).

Python libraries suitable for Machine Learning.

Python machine learning libraries have grown to become the most preferred language for machine learning algorithm implementations. Let's have a look at the main Python libraries used for machine learning.

Top Python Machine Learning Libraries

1) NumPy

NumPy is a well known general-purpose array-processing package. An extensive collection of high complexity mathematical functions make NumPy powerful to process large multi-dimensional arrays and matrices. NumPy is very useful for handling linear algebra, Fourier transforms, and random numbers. Other libraries like TensorFlow uses NumPy at the backend for manipulating tensors.

With NumPy, you can define arbitrary data types and easily integrate with most databases. NumPy can also serve as an efficient multi-dimensional container for any generic data that is in any datatype. The key features of NumPy include powerful N-dimensional array object, broadcasting functions, and out-of-box tools to integrate C/C++ and Fortran code.

2) SciPy

With machine learning growing at supersonic speed, many Python developers were creating **python libraries for machine learning**, especially for scientific and analytical computing. Travis Oliphant, Eric Jones, and Pearu Peterson in 2001 decided to merge most of these bits and pieces codes and standardize it. The resulting library was then named as SciPy library.

The current development of the SciPy library is supported and sponsored by an open community of developers and distributed under the free BSD license.

The SciPy library offers modules for linear algebra, image optimization, integration interpolation, special functions, Fast Fourier transform, signal and image processing, Ordinary Differential Equation (ODE) solving, and other computational tasks in science and analytics.

The underlying data structure used by SciPy is a multi-dimensional array provided by the NumPy module. SciPy depends on NumPy for the array manipulation subroutines. The SciPy library was built to work with NumPy arrays along with providing user-friendly and efficient numerical functions.

3) Scikit-learn

In 2007, **David Cournapeau** developed the Scikit-learn library as part of the Google Summer of Code project. In 2010 INRIA involved and did the public release in January 2010. Skikit-learn was built on top of two Python libraries – NumPy and SciPy and has become the most popular Python machine learning library for developing machine learning algorithms.

Scikit-learn has a wide range of supervised and unsupervised learning algorithms that works on a consistent interface in Python. The library can also be used for data-mining and data analysis. The main machine learning functions that the Scikit-learn library can handle are classification, regression, clustering, dimensionality reduction, model selection, and preprocessing.

4) Theano

Theano is a **python machine learning library** that can act as an optimizing compiler for evaluating and manipulating mathematical expressions and matrix calculations. Built on

NumPy, Theano exhibits a tight integration with NumPy and has a very similar interface. Theano can work on Graphics Processing Unit (GPU) and CPU.
Working on GPU architecture yields faster results. Theano can perform data-intensive computations up to 140x faster on GPU than on a CPU. Theano can automatically avoid errors and bugs when dealing with logarithmic and exponential functions. Theano has built-in tools for unit-testing and validation, thereby avoiding bugs and problems.

5) TensorFlow

TensorFlow was developed for Google's internal use by the Google Brain team. Its first release came in November 2015 under Apache License 2.0. TensorFlow is a popular computational framework for creating **machine learning models**. TensorFlow supports a variety of different toolkits for constructing models at varying levels of abstraction.
TensorFlow exposes a very stable Python and C++ APIs. It can expose, backward compatible APIs for other languages too, but they might be unstable. TensorFlow has a flexible architecture with which it can run on a variety of computational platforms CPUs, GPUs, and TPUs. TPU stands for Tensor processing unit, a hardware chip built around TensorFlow for machine learning and artificial intelligence.

6) Keras

Keras has over 200,000 users as of November 2017. Keras is an open-source library used for neural networks and machine learning. Keras can run on top of TensorFlow, Theano, Microsoft Cognitive Toolkit, R, or PlaidML. Keras also can run efficiently on CPU and GPU.
Keras works with neural-network building blocks like layers, objectives, activation functions, and optimizers. Keras also have a bunch of features to work on images and text images that comes handy when writing Deep Neural Network code.
Apart from the standard neural network, Keras supports convolutional and recurrent neural networks.

7) PyTorch

PyTorch has a range of tools and libraries that support computer vision, machine learning, and natural language processing. The PyTorch library is open-source and is based on the Torch library. The most significant advantage of PyTorch library is it's ease of learning and using.
PyTorch can smoothly integrate with the python data science stack, including NumPy. You will hardly make out a difference between NumPy and PyTorch. PyTorch also allows developers to perform computations on Tensors. PyTorch has a robust framework to build computational graphs on the go and even change them in runtime. Other advantages of PyTorch include multi GPU support, simplified preprocessors, and custom data loaders.

8) Pandas

Pandas are turning up to be the most popular Python library that is used for data analysis with support for fast, flexible, and expressive data structures designed to work on both "relational" or "labeled" data. Pandas today is an inevitable library for solving practical, real-world data analysis in Python. Pandas is highly stable, providing highly optimized performance. The backend code is purely written in C or Python.

The two main types of data structures used by pandas are :

Series (1-dimensional)

DataFrame (2-dimensional)

These two put together can handle a vast majority of data requirements and use cases from most sectors like science, statistics, social, finance, and of course, analytics and other areas of engineering.

Pandas support and perform well with different kinds of data including the below :

Tabular data with columns of heterogeneous data. For instance, consider the data coming from the SQL table or Excel spreadsheet.
Ordered and unordered time series data. The frequency of time series need not be fixed, unlike other libraries and tools. Pandas is exceptionally robust in handling uneven time-series data
Arbitrary matrix data with the homogeneous or heterogeneous type of data in the rows and columns

Any other form of statistical or observational data sets. The data need not be labeled at all. Pandas data structure can process it even without labeling.

9) Matplotlib

Matplotlib is a data visualization library that is used for 2D plotting to produce publication-quality image plots and figures in a variety of formats. The library helps to generate histograms, plots, error charts, scatter plots, bar charts with just a few lines of code.

It provides a MATLAB-like interface and is exceptionally user-friendly. It works by using standard GUI toolkits like GTK+, wxPython, Tkinter, or Qt to provide an object-oriented API that helps programmers to embed graphs and plots into their applications.

TEXT / REFERENCE BOOKS

1. Chris Albon : Machine Learning with Python Cookbook , O"Reilly Media, Inc.2018
2. Stephen Marsland, "Machine Learning – An Algorithmic Perspective", Second Edition, Chapman and Hall/CRC Machine Learning and Pattern Recognition Series, 2014
3. Tom M. Mitchell, Machine Learning, India Edition 2013, McGraw Hill Education
4. Machine Learning: The art and Science of algorithms that make sense of data, Peter Flach, Cambridge University Press, 2012
5. EthemAlpaydın, Introduction to machine learning, second edition, MIT press.
6. T. Hastie, R. Tibshirani and J. Friedman, "Elements of Statistical Learning", Springer Series , 2nd edition
7. Sebastian Raschka, "Python Machine Learning",Second Edition.Packt Publication

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT- II CLASSIFIERS

UNIT II CLASSIFIERS

Classification, K- nearest neighbour, Decision Trees, Implementing Decision Tree, building a Tree, Random Forests - Working of Random Forest, Pros and Cons of Random Forest, Naiver Bayes, building model Using Naiver Bayes.

KNN:

In statistics, the k-nearest neighbors algorithm (k-NN) is a non parametric classification method first developed by Evelyn Fix and Joseph Hodges in 1951 and later expanded by Thomas Cover.

It is used for classification and regression. In both cases, the input consists of the k closest training examples in data set. The output depends on whether k-NN is used for classification or regression.

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm.

It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well −

- **Lazy learning algorithm** − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
- **Non-parametric learning algorithm** − KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

Working of KNN Algorithm

K-nearest neighbors (KNN) algorithm uses 'feature similarity' to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps −

Step 1 − For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step 2 − Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3 − For each point in the test data do the following −

3.1 − Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.

3.2 − Now, based on the distance value, sort them in ascending order.

3.3 − Next, it will choose the top K rows from the sorted array.

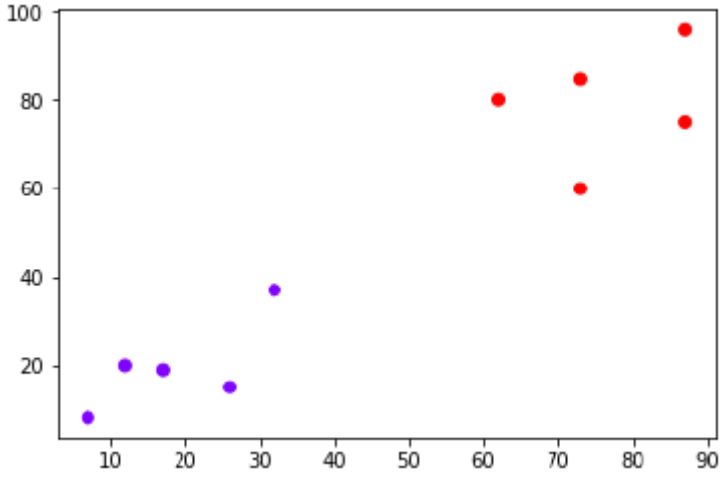
3.4 − Now, it will assign a class to the test point based on most frequent class of these rows.

Step 4 − End

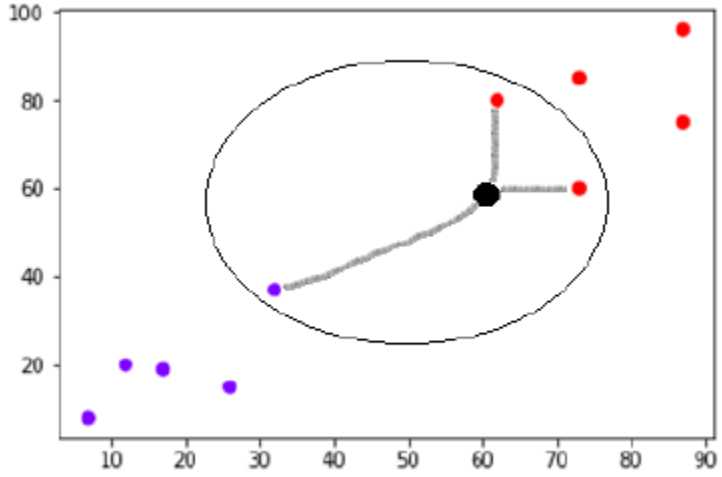
Example

The following is an example to understand the concept of K and working of KNN algorithm −

Suppose we have a dataset which can be plotted as follows −



Now, we need to classify new data point with black dot (at point 60,60) into blue or red class. We are assuming K = 3 i.e. it would find three nearest data points. It is shown in the next diagram −



We can see in the above diagram the three nearest neighbors of the data point with black dot. Among those three, two of them lies in Red class hence the black dot will also be assigned in red class.

Pros and Cons of KNN

Pros

- It is very simple algorithm to understand and interpret.
- It is very useful for nonlinear data because there is no assumption about data in this algorithm.
- It is a versatile algorithm as we can use it for classification as well as regression.
- It has relatively high accuracy but there are much better supervised learning models than KNN.

Cons

- It is computationally a bit expensive algorithm because it stores all the training data.
- High memory storage required as compared to other supervised learning algorithms.
- Prediction is slow in case of big N.
- It is very sensitive to the scale of data as well as irrelevant features.

Applications of KNN

The following are some of the areas in which KNN can be applied successfully −

Banking System

KNN can be used in banking system to predict weather an individual is fit for loan approval? Does that individual have the characteristics similar to the defaulters one?

Calculating Credit Ratings

KNN algorithms can be used to find an individual's credit rating by comparing with the persons having similar traits.

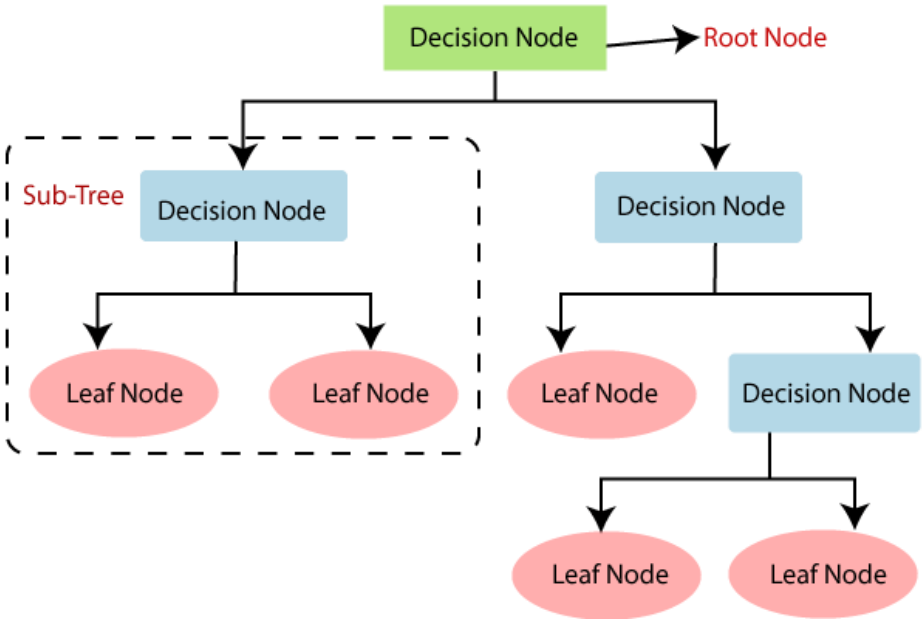
Politics

With the help of KNN algorithms, we can classify a potential voter into various classes like "Will Vote", "Will not Vote", "Will Vote to Party 'Congress', "Will Vote to Party 'BJP'.

Other areas in which KNN algorithm can be used are Speech Recognition, Handwriting Detection, Image Recognition and Video Recognition.

Decision Trees

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- **It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.**
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.



Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

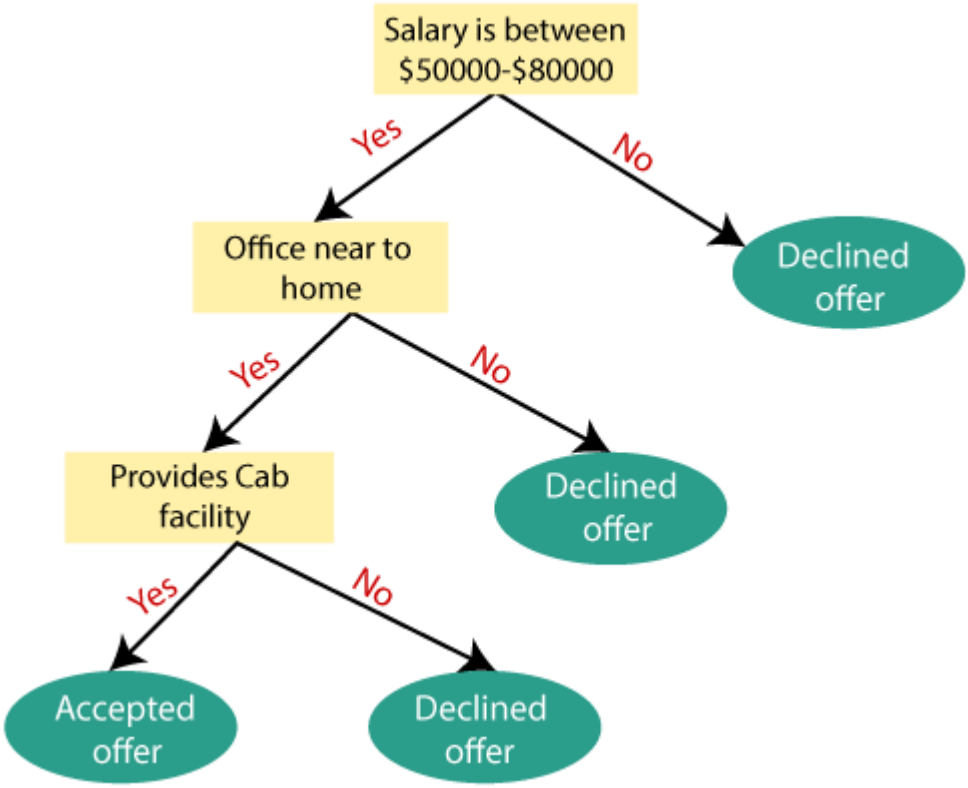
How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

1. Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

Gini Index= $1- \sum_j P_j^2$

Pruning: Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

- **Cost Complexity Pruning**
- **Reduced Error Pruning.**

Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.

- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

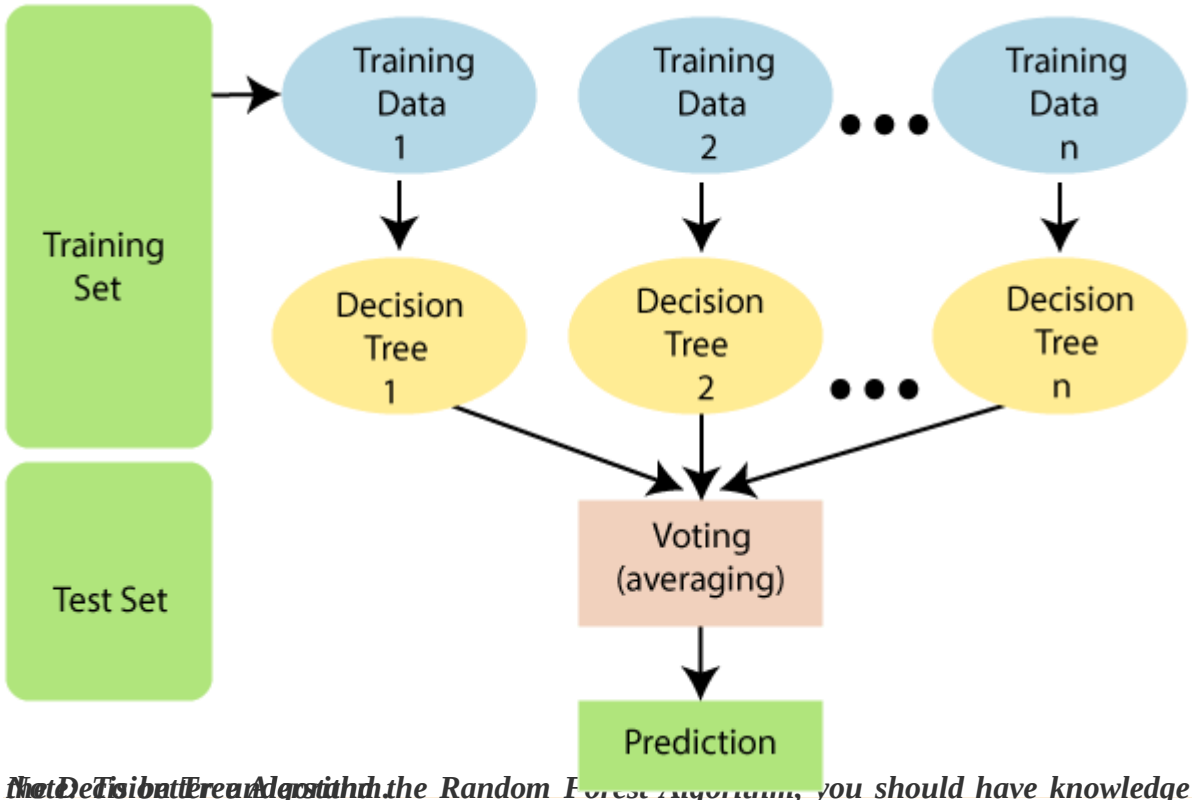
Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:



Note: To understand the Random Forest Algorithm, you should have knowledge of the Decision Tree Algorithm.

Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

Why use Random Forest?

Below are some points that explain why we should use the Random Forest algorithm:

<=”” li=””>

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

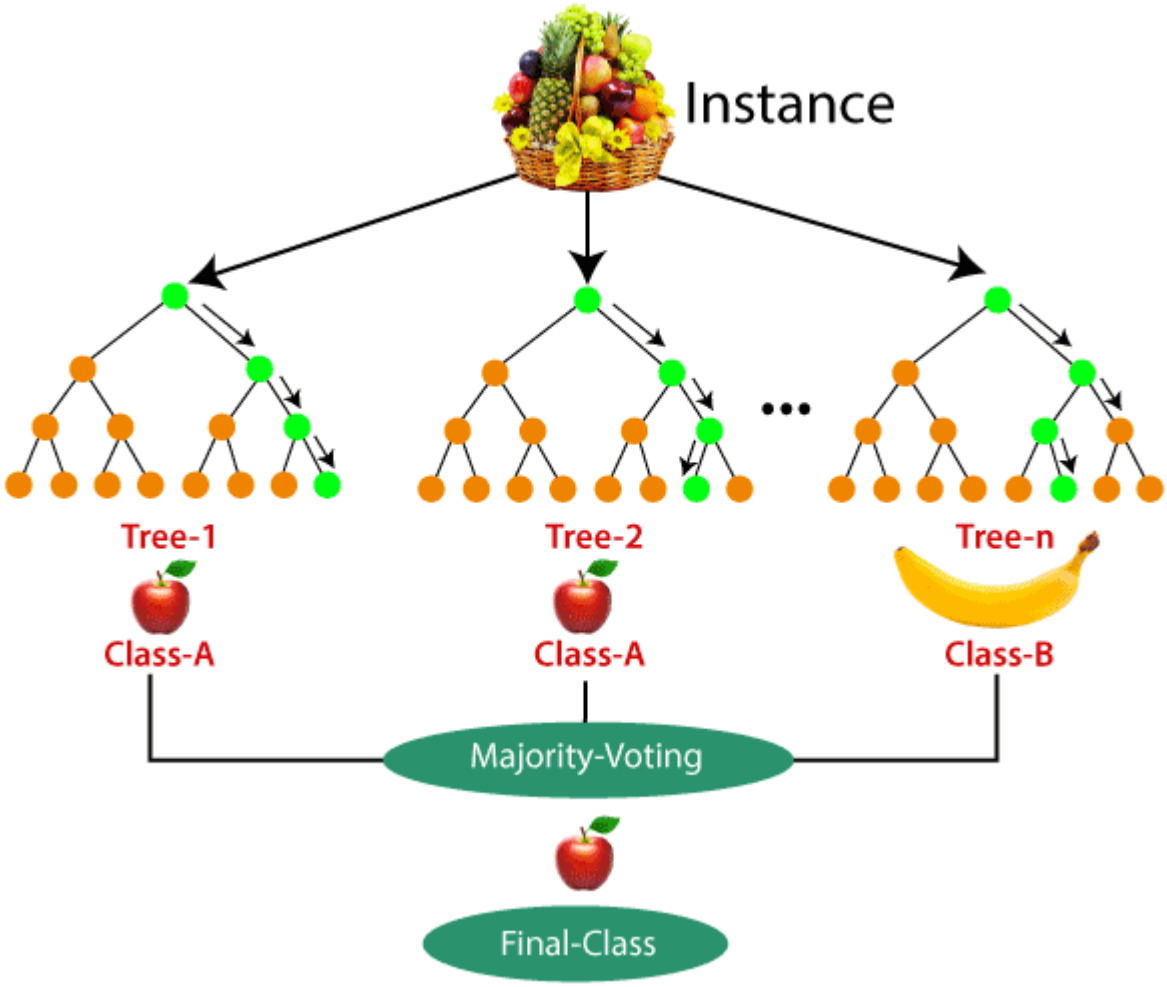
Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

The working of the algorithm can be better understood by the below example:

Example: Suppose there is a dataset that contains multiple fruit images. So, this dataset is given to the Random forest classifier. The dataset is divided into subsets and given to each decision tree. During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on the majority of results, the Random Forest classifier predicts the final decision. Consider the below image:



Applications of Random Forest

There are mainly four sectors where Random forest mostly used:

1. **Banking:** Banking sector mostly uses this algorithm for the identification of loan risk.
2. **Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.
3. **Land Use:** We can identify the areas of similar land use by this algorithm.
4. **Marketing:** Marketing trends can be identified using this algorithm.

Advantages of Random Forest

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

Disadvantages of Random Forest

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

Naiver Bayes

Bayesian Decision Theory

Bayesian framework assumes that we always have a prior distribution for everything.

- The prior may be very vague.
- When we see some data, we combine our prior distribution with a likelihood termto get a posterior distribution.
- The likelihood term takes into account how probable the observed data is giventhe parameters of the model.
  - It favors parameter settings that make the data likely.
  - It fights the prior
  - With enough data the likelihood terms always win.

$P(H|\boldsymbol{X})$ is the **posterior probability**, or *a posteriori probability*, of H conditioned on $\boldsymbol{X}$. For example, suppose our world of data tuples is confined to customers described by the attributes *age* and *income*, respectively, and that $\boldsymbol{X}$ is a 35-year-old customer with an income of $40,000. Suppose that H is the hypothesis that our customer will buy a computer. Then $P(H|\boldsymbol{X})$ reflects the probability that customer $\boldsymbol{X}$ will buy a computer given that we know the customer's age and income.

In contrast, $P(H)$ is the **prior probability**, or *a priori probability*, of H. For our example, this is the probability that any given customer will buy a computer, regardless of age, income, or any other information, for that matter.

Bayes' theorem is useful in that it provides a way of calculating the posterior probability $P(H|\boldsymbol{X})$, from $P(H)$, $P(\boldsymbol{X}|H)$, and $P(\boldsymbol{X})$.

Bayes' theorem is

$$P(H|\boldsymbol{X}) = \frac{P(\boldsymbol{X}|H)P(H)}{P(\boldsymbol{X})}.$$

Naïve Bayesian Classification

The **naïve Bayesian** classifier, or **simple Bayesian** classifier, works as follows:

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, $X = (x_1, x_2, \ldots, x_n)$, depicting n measurements made on the tuple from n attributes, respectively, $A_1, A_2, \ldots, A_n$.

2. Suppose that there are m classes, $C_1, C_2, \ldots, C_m$. Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. That is, the naïve Bayesian classifier predicts that tuple X belongs to the class C_i if and only if

$$P(C_i|X) > P(C_j|X) \quad \text{for } 1 \leq j \leq m, j \neq i.$$

   Thus, we maximize $P(C_i|X)$. The class C_i for which $P(C_i|X)$ is maximized is called the *maximum posteriori hypothesis*. By Bayes' theorem (Eq. 8.10),

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}.$$

3. As $P(X)$ is constant for all classes, only $P(X|C_i)P(C_i)$ needs to be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is, $P(C_1) = P(C_2) = \cdots = P(C_m)$, and we would therefore maximize $P(X|C_i)$. Otherwise, we maximize $P(X|C_i)P(C_i)$. Note that the class prior probabilities may be estimated by $P(C_i) = |C_{i,D}|/|D|$, where $|C_{i,D}|$ is the number of training tuples of class C_i in D.

4. Given data sets with many attributes, it would be extremely computationally expensive to compute $P(X|C_i)$. To reduce computation in evaluating $P(X|C_i)$, the naïve assumption of **class-conditional independence** is made. This presumes that the attributes' values are conditionally independent of one another, given the class label of the tuple (i.e., that there are no dependence relationships among the attributes). Thus,

$$\begin{aligned} P(X|C_i) &= \prod_{k=1}^{n} P(x_k|C_i) \\ &= P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i). \end{aligned}$$

   We can easily estimate the probabilities $P(x_1|C_i), P(x_2|C_i), \ldots, P(x_n|C_i)$ from the training tuples. Recall that here x_k refers to the value of attribute A_k for tuple X. For each attribute, we look at whether the attribute is categorical or continuous-valued. For instance, to compute $P(X|C_i)$, we consider the following:

(a) If A_k is categorical, then $P(x_k|C_i)$ is the number of tuples of class C_i in D having the value x_k for A_k, divided by $|C_{i,D}|$, the number of tuples of class C_i in D.

(b) If A_k is continuous-valued, then we need to do a bit more work, but the calculation is pretty straightforward. A continuous-valued attribute is typically assumed to have a Gaussian distribution with a mean μ and standard deviation σ, defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

so that

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}).$$

Given database:

Class-Labeled Training Tuples from the *AllElectronics* Customer Database

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Example:

Predicting a class label using naïve Bayesian classification. We wish to predict the class label of a tuple using naïve Bayesian classification, given the same training data as in Example 8.3 for decision tree induction. The training data were shown earlier in Table 8.1. The data tuples are described by the attributes *age*, *income*, *student*, and *credit_rating*. The class label attribute, *buys_computer*, has two distinct values (namely, {*yes*, *no*}). Let C_1 correspond to the class *buys_computer* = *yes* and C_2 correspond to *buys_computer* = *no*. The tuple we wish to classify is

$$\boldsymbol{X} = (age = youth,\ income = medium,\ student = yes,\ credit\_rating = fair)$$

We need to maximize $P(\boldsymbol{X}|C_i)P(C_i)$, for $i = 1, 2$. $P(C_i)$, the prior probability of each class, can be computed based on the training tuples:

$$P(buys\_computer = yes) = 9/14 = 0.643$$
$$P(buys\_computer = no) = 5/14 = 0.357$$

To compute $P(\boldsymbol{X}|C_i)$, for $i = 1, 2$, we compute the following conditional probabilities:

$$P(age = youth \mid buys\_computer = yes) = 2/9 = 0.222$$
$$P(age = youth \mid buys\_computer = no) = 3/5 = 0.600$$
$$P(income = medium \mid buys\_computer = yes) = 4/9 = 0.444$$
$$P(income = medium \mid buys\_computer = no) = 2/5 = 0.400$$
$$P(student = yes \mid buys\_computer = yes) = 6/9 = 0.667$$
$$P(student = yes \mid buys\_computer = no) = 1/5 = 0.200$$
$$P(credit\_rating = fair \mid buys\_computer = yes) = 6/9 = 0.667$$
$$P(credit\_rating = fair \mid buys\_computer = no) = 2/5 = 0.400$$

Using these probabilities, we obtain

$$\begin{aligned} P(\boldsymbol{X}|buys\_computer = yes) &= P(age = youth \mid buys\_computer = yes) \\ &\quad \times P(income = medium \mid buys\_computer = yes) \\ &\quad \times P(student = yes \mid buys\_computer = yes) \\ &\quad \times P(credit\_rating = fair \mid buys\_computer = yes) \\ &= 0.222 \times 0.444 \times 0.667 \times 0.667 = 0.044. \end{aligned}$$

Similarly,

$$P(\boldsymbol{X}|buys\_computer = no) = 0.600 \times 0.400 \times 0.200 \times 0.400 = 0.019.$$

To find the class, C_i, that maximizes $P(\boldsymbol{X}|C_i)P(C_i)$, we compute

$$P(\boldsymbol{X}|buys\_computer = yes)P(buys\_computer = yes) = 0.044 \times 0.643 = 0.028$$
$$P(\boldsymbol{X}|buys\_computer = no)P(buys\_computer = no) = 0.019 \times 0.357 = 0.007$$

Therefore, the naïve Bayesian classifier predicts *buys_computer* = *yes* for tuple $\boldsymbol{X}$.

Losses and Risks:

- Actions: α_i
- Loss of α_i when the state is C_k : λ_{ik}
- Expected risk (Duda and Hart, 1973)

$$R(\alpha_i \mid \mathbf{x}) = \sum_{k=1}^{K} \lambda_{ik} P(C_k \mid \mathbf{x})$$

$$\text{choose } \alpha_i \text{ if } R(\alpha_i \mid \mathbf{x}) = \min_k R(\alpha_k \mid \mathbf{x})$$

0/1 Loss

$$\lambda_{ik} = \begin{cases} 0 \text{ if } i = k \\ 1 \text{ if } i \neq k \end{cases}$$

$$\begin{aligned} R(\alpha_i \mid \mathbf{x}) &= \sum_{k=1}^{K} \lambda_{ik} P(C_k \mid \mathbf{x}) \\ &= \sum_{k \neq i} P(C_k \mid \mathbf{x}) \\ &= 1 - P(C_i \mid \mathbf{x}) \end{aligned}$$

For minimum risk, choose the most probable class

Reject

$$\lambda_{ik} = \begin{cases} 0 & \text{if } i = k \\ \lambda & \text{if } i = K+1, \quad 0 < \lambda < 1 \\ 1 & \text{otherwise} \end{cases}$$

$$R(\alpha_{K+1} \mid \mathbf{x}) = \sum_{k=1}^{K} \lambda P(C_k \mid \mathbf{x}) = \lambda$$

$$R(\alpha_i \mid \mathbf{x}) = \sum_{k \neq i} P(C_k \mid \mathbf{x}) = 1 - P(C_i \mid \mathbf{x})$$

$$\begin{aligned} &\text{choose } C_i \quad \text{if } P(C_i \mid \mathbf{x}) > P(C_k \mid \mathbf{x}) \ \forall k \neq i \text{ and } P(C_i \mid \mathbf{x}) > 1 - \lambda \\ &\text{reject} \qquad \text{otherwise} \end{aligned}$$

Discriminant Functions

Classification can also be seen as implementing a set of *discriminant f*unctions, *gi(x), i* = 1, . . *K*, such that we

$$\text{choose } C_i \text{ if } g_i(\boldsymbol{x}) = \max_k g_k(\boldsymbol{x})$$

$$g_i(\boldsymbol{x}) = \begin{cases} -R(\alpha_i \mid \boldsymbol{x}) \\ P(C_i \mid \boldsymbol{x}) \\ p(\boldsymbol{x} \mid C_i)P(C_i) \end{cases}$$

$$K \text{ decision regions} \mathcal{R}_1,\ldots,\mathcal{R}_K$$

$$\mathcal{R}_i = \{\boldsymbol{x} \mid g_i(\boldsymbol{x}) = \max_k g_k(\boldsymbol{x})\}$$

- Dichotomizer (K=2) vs Polychotomizer (K>2)
- $g(\boldsymbol{x}) = g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$

$$\text{choose} \begin{cases} C_1 \text{ if } g(\boldsymbol{x}) > 0 \\ C_2 \text{ otherwise} \end{cases}$$

- *Log odds:*

$$\log \frac{P(C_1 \mid \boldsymbol{x})}{P(C_2 \mid \boldsymbol{x})}$$

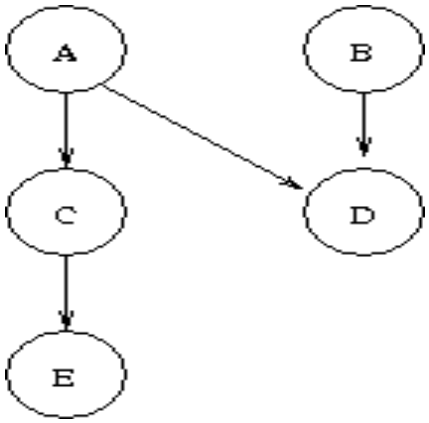
Building model Using Naiver Bayes

Bayesian networks

A Bayesian network, Bayes network, belief network, Bayes(ian) model or probabilistic directed acyclic graphical model is a probabilistic graphical model (a type of statistical model) that represents a set of variables and their conditional dependencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Bayesian Net Example:

Consider the following Bayesian network:



Thus, the independence expressed in this Bayesian net are thatA and B are (absolutely) independent.
C is independent of B given A.
D is independent of C given A and B.
E is independent of A, B, and D given C.

Suppose that the net further records the following probabilities:
Prob(A=T) = 0.3
Prob(B=T) = 0.6
Prob(C=T|A=T) = 0.8
Prob(C=T|A=F) = 0.4
Prob(D=T|A=T,B=T) = 0.7
Prob(D=T|A=T,B=F) = 0.8
Prob(D=T|A=F,B=T) = 0.1
Prob(D=T|A=F,B=F) = 0.2
Prob(E=T|C=T) = 0.7
Prob(E=T|C=F) = 0.2

Some sample computations:

Prob(D=T):

P(D=T) =

P(D=T,A=T,B=T) + P(D=T,A=T,B=F) + P(D=T,A=F,B=T) + P(D=T,A=F,B=F) =

P(D=T|A=T,B=T) P(A=T,B=T) + P(D=T|A=T,B=F) P(A=T,B=F) +P(D=T|A=F,B=T) P(A=F,B=T) + P(D=T|A=F,B=F) P(A=F,B=F) =
(since A and B are independent absolutely)

P(D=T|A=T,B=T) P(A=T) P(B=T) + P(D=T|A=T,B=F) P(A=T) P(B=F) +P(D=T|A=F,B=T) P(A=F) P(B=T) + P(D=T|A=F,B=F) P(A=F) P(B=F) =

0.7*0.3*0.6 + 0.8*0.3*0.4 + 0.1*0.7*0.6 + 0.2*0.7*0.4 = 0.32

Prob(A=T|C=T):

P(A=T|C=T) = P(C=T|A=T)P(A=T) / P(C=T).

Now, P(C=T) = P(C=T,A=T) + P(C=T,A=F) =
P(C=T|A=T)P(A=T) + P(C=T|A=F)P(A=F) =
0.8*0.3+ 0.4*0.7 = 0.52

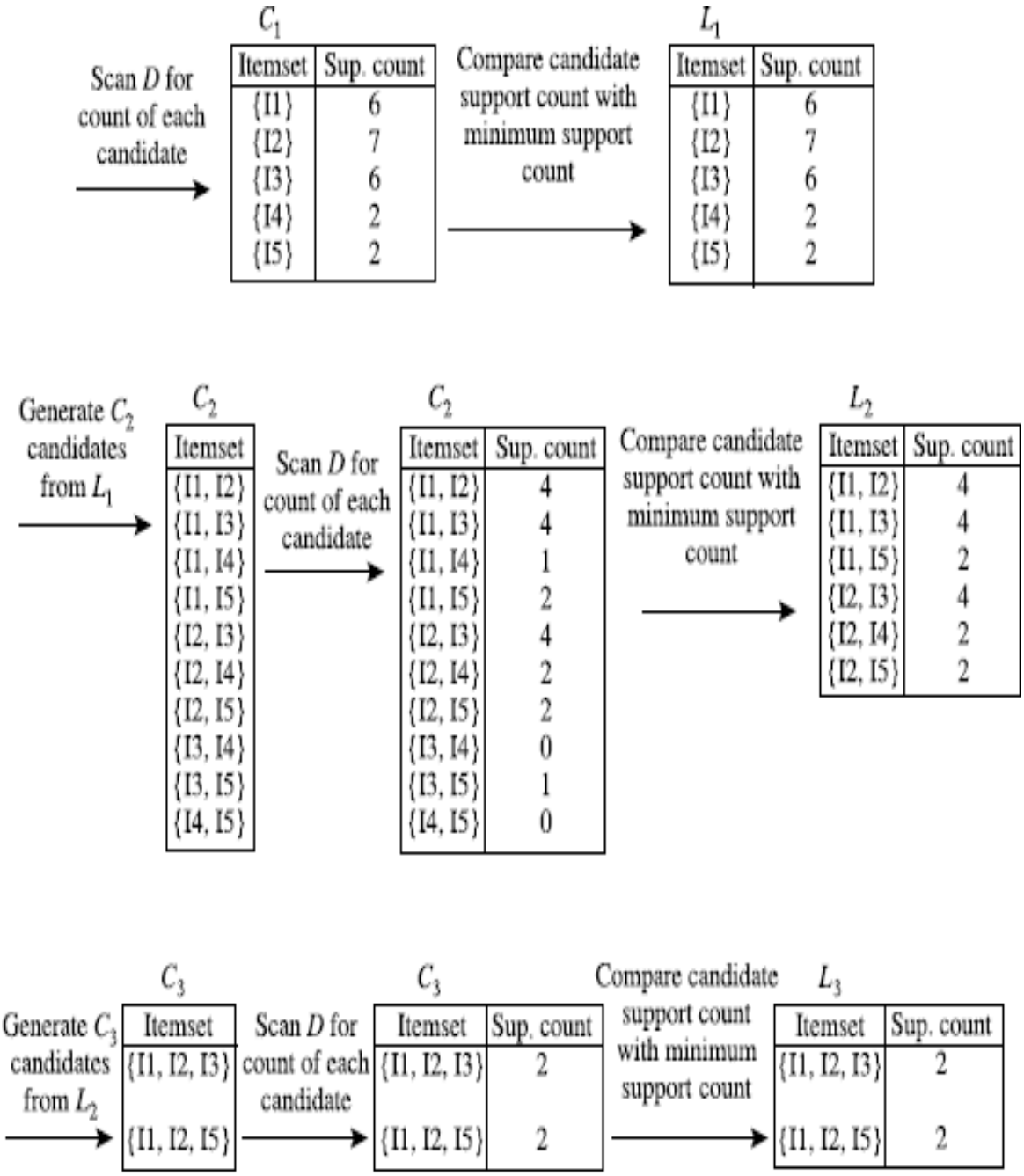
So P(C=T|A=T)P(A=T) / P(C=T) = 0.8*0.3/0.52= 0.46.

Association rule

Association rule mining is explained using the **Apriori Algorithm**.

Transactional Data for an *AllElectronics* Branch

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3



Generation of the candidate itemsets and frequent itemsets, where the minimum support count is 2.

Apriori Algorithm:

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D, a database of transactions;
- *min_sup*, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)   L1 = find_frequent_1-itemsets(D);
(2)   for (k = 2; Lk−1 ≠ φ; k++) {
(3)       Ck = apriori_gen(Lk−1);
(4)       for each transaction t ∈ D { // scan D for counts
(5)           Ct = subset(Ck, t); // get the subsets of t that are candidates
(6)           for each candidate c ∈ Ct
(7)               c.count++;
(8)       }
(9)       Lk = {c ∈ Ck | c.count ≥ min_sup}
(10)  }
(11)  return L = ∪k Lk;

procedure apriori_gen(Lk−1:frequent (k − 1)-itemsets)
(1)   for each itemset l1 ∈ Lk−1
(2)       for each itemset l2 ∈ Lk−1
(3)           if (l1[1] = l2[1]) ∧ (l1[2] = l2[2])
                  ∧... ∧ (l1[k − 2] = l2[k − 2]) ∧ (l1[k − 1] < l2[k − 1]) then {
(4)               c = l1 ⋈ l2; // join step: generate candidates
(5)               if has_infrequent_subset(c, Lk−1) then
(6)                   delete c; // prune step: remove unfruitful candidate
(7)               else add c to Ck;
(8)           }
(9)   return Ck;

procedure has_infrequent_subset(c: candidate k-itemset;
          Lk−1: frequent (k − 1)-itemsets); // use prior knowledge
(1)   for each (k − 1)-subset s of c
(2)       if s ∉ Lk−1 then
(3)           return TRUE;
(4)   return FALSE;
```

Confidence:

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support\_count(A \cup B)}{support\_count(A)}.$$

The conditional probability is expressed in terms of itemset support count, where $support\_count(A \cup B)$ is the number of transactions containing the itemsets $A \cup B$, and $support\_count(A)$ is the number of transactions containing the itemset A.

Generating association rules. Let's try an example based on the transactional data for *AllElectronics* shown before in Table 6.1. The data contain frequent itemset $X = \{I1, I2, I5\}$. What are the association rules that can be generated from X? The nonempty subsets of X are {I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, and {I5}. The resulting association rules are as shown below, each listed with its confidence:

$\{I1, I2\} \Rightarrow I5$, $confidence = 2/4 = 50\%$
$\{I1, I5\} \Rightarrow I2$, $confidence = 2/2 = 100\%$
$\{I2, I5\} \Rightarrow I1$, $confidence = 2/2 = 100\%$
$I1 \Rightarrow \{I2, I5\}$, $confidence = 2/6 = 33\%$
$I2 \Rightarrow \{I1, I5\}$, $confidence = 2/7 = 29\%$
$I5 \Rightarrow \{I1, I2\}$, $confidence = 2/2 = 100\%$

If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules are output, because these are the only ones generated that are strong.

Parametric Methods

Parametric Estimation

- X = $\{ x^t \}_t$ where $x^t \sim p(x)$
- Parametric estimation:

Assume a form for $p(x \mid \theta)$ and estimate θ, its sufficient statistics,using X

e.g., N (μ, σ^2) where $\theta = \{ \mu, \sigma^2 \}$

Maximum Likelihood Estimation:

- Likelihood of θ given the sample X

$l(\theta|X) = p(X|\theta) = \prod_t p(x^t|\theta)$

- Log likelihood

$L(\theta|X) = \log l(\theta|X) = \sum_t \log p(x^t|\theta)$

- Maximum likelihood estimator (MLE)

$\theta^* = \text{argmax}_\theta L(\theta|X)$

Examples: Bernoulli/Multinomial:

- Bernoulli: Two states, failure/success, x in {0,1}

$$P(x) = p_o^x (1 - p_o)^{(1-x)}$$

$$\mathcal{L}(p_o|\mathcal{X}) = \log \Pi_t p_o^{x^t} (1 - p_o)^{(1-x^t)}$$

$$\text{MLE: } p_o = \Sigma_t x^t / N$$

- Multinomial: $K>2$ states, x_i in {0,1}

$$P(x_1, x_2, \ldots, x_K) = \Pi_i p_i^{x_i}$$

$$\mathcal{L}(p_1, p_2, \ldots, p_K|\mathcal{X}) = \log \Pi_t \Pi_i p_i^{x_i^t}$$

$$\text{MLE: } p_i = \Sigma_t x_i^t / N$$

Gaussian (Normal) Distribution:

- $p(x) = \mathcal{N}(\mu, \sigma^2)$

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

- MLE for μ and σ^2:

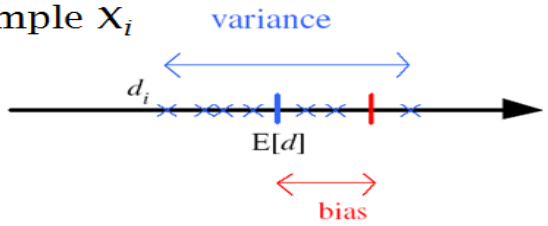
$$m = \frac{\sum_t x^t}{N}$$

$$s^2 = \frac{\sum_t (x^t - m)^2}{N}$$

Bias and Variance:

Unknown parameter θ
Estimator $d_i = d(X_i)$ on sample X_i

Bias: $b_\theta(d) = E[d] - \theta$
Variance: $E[(d-E[d])^2]$



Mean square error:
$r(d,\theta) = E[(d-\theta)^2]$
$= (E[d] - \theta)^2 + E[(d-E[d])^2]$
$=$ Bias^2 + Variance

Classification

$$g_i(x) = p(x \mid C_i)P(C_i)$$
or equivalently
$$g_i(x) = \log p(x \mid C_i) + \log P(C_i)$$

$$p(x \mid C_i) = \frac{1}{\sqrt{2\pi}\sigma_i}\exp\left[-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right]$$

$$g_i(x) = -\frac{1}{2}\log 2\pi - \log \sigma_i - \frac{(x-\mu_i)^2}{2\sigma_i^2} + \log P(C_i)$$

- Given the sample $\mathcal{X} = \{x^t, r^t\}_{t=1}^N$

$$x \in \Re \qquad r_i^t = \begin{cases} 1 \text{ if } x^t \in C_i \\ 0 \text{ if } x^t \in C_j, j \neq i \end{cases}$$

- ML estimates are

$$\hat{P}(C_i) = \frac{\sum_t r_i^t}{N} \quad m_i = \frac{\sum_t x^t r_i^t}{\sum_t r_i^t} \quad s_i^2 = \frac{\sum_t (x^t - m_i)^2 r_i^t}{\sum_t r_i^t}$$

- Discriminant becomes

$$g_i(x) = -\frac{1}{2}\log 2\pi - \log s_i - \frac{(x-m_i)^2}{2s_i^2} + \log \hat{P}(C_i)$$

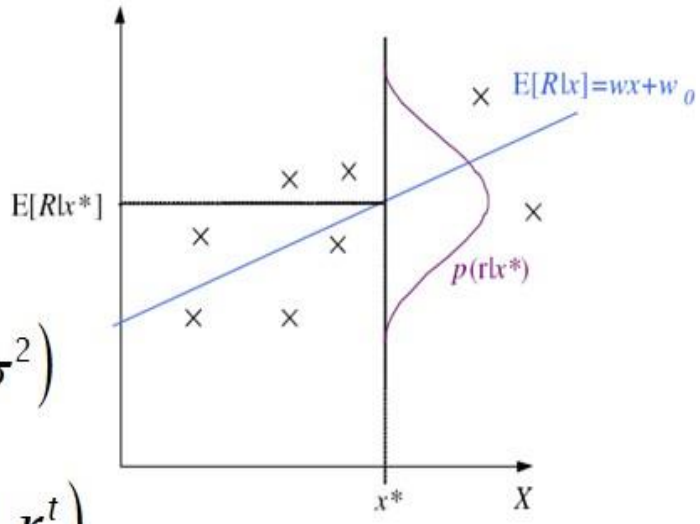
Regression

$$r = f(x) + \varepsilon$$

$$\text{estimator}: g(x \mid \theta)$$

$$\varepsilon \sim \mathcal{N}(0, \sigma^2)$$

$$p(r \mid x) \sim \mathcal{N}(g(x \mid \theta), \sigma^2)$$



$$\mathcal{L}(\theta \mid \mathcal{X}) = \log \prod_{t=1}^{N} p(x^t, r^t)$$

$$= \log \prod_{t=1}^{N} p(r^t \mid x^t) + \log \prod_{t=1}^{N} p(x^t)$$

$$\mathcal{L}(\theta \mid \mathcal{X}) = \log \prod_{t=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{\left[r^t - g(x^t \mid \theta)\right]^2}{2\sigma^2} \right]$$

$$= -N \log \sqrt{2\pi}\sigma - \frac{1}{2\sigma^2} \sum_{t=1}^{N} \left[r^t - g(x^t \mid \theta)\right]^2$$

$$E(\theta \mid \mathcal{X}) = \frac{1}{2} \sum_{t=1}^{N} \left[r^t - g(x^t \mid \theta)\right]^2$$

Linear Regression:

$$g(x^t \mid w_1, w_0) = w_1 x^t + w_0$$

$$\sum_t r^t = N w_0 + w_1 \sum_t x^t$$

$$\sum_t r^t x^t = w_0 \sum_t x^t + w_1 \sum_t (x^t)^2$$

$$\mathbf{A} = \begin{bmatrix} N & \sum_t x^t \\ \sum_t x^t & \sum_t (x^t)^2 \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} \sum_t r^t \\ \sum_t r^t x^t \end{bmatrix}$$

$$\mathbf{w} = \mathbf{A}^{-1}\mathbf{y}$$

Other Error Measures:

- Square Error: $E(\theta \mid \mathcal{X}) = \frac{1}{2}\sum_{t=1}^{N}\left[r^t - g(x^t \mid \theta)\right]^2$
- Relative Square Error: $E(\theta \mid \mathcal{X}) = \frac{\sum_{t=1}^{N}\left[r^t - g(x^t \mid \theta)\right]^2}{\sum_{t=1}^{N}\left[r^t - \bar{r}\right]^2}$
- Absolute Error: $E(\theta|\mathcal{X}) = \sum_t |r^t - g(x^t|\theta)|$
- ε-sensitive Error:

$$E(\theta|\mathcal{X}) = \sum_t 1(|r^t - g(x^t|\theta)|>\varepsilon)\ (|r^t - g(x^t|\theta)| - \varepsilon)$$

Multivariate Methods

Data:

- Multiple measurements (sensors)
- d inputs/features/attributes: d-variate
- N instances/observations/examples

$$\mathbf{X} = \begin{bmatrix} X_1^1 & X_2^1 & \cdots & X_d^1 \\ X_1^2 & X_2^2 & \cdots & X_d^2 \\ \vdots & & & \\ X_1^N & X_2^N & \cdots & X_d^N \end{bmatrix}$$

Multivariate Parameters:

$$\text{Mean} : E[\boldsymbol{x}] = \boldsymbol{\mu} = [\mu_1, ..., \mu_d]^T$$

$$\text{Covariance} : \sigma_{ij} \equiv \text{Cov}(X_i, X_j)$$

$$\text{Correlation} : \text{Corr}(X_i, X_j) \equiv \rho_{ij} = \frac{\sigma_{ij}}{\sigma_i \sigma_j}$$

$$\Sigma \equiv \text{Cov}(\boldsymbol{X}) = E\left[(\boldsymbol{X} - \mu)(\boldsymbol{X} - \mu)^T\right] = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1d} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2d} \\ \vdots & & & \\ \sigma_{d1} & \sigma_{d2} & \cdots & \sigma_d^2 \end{bmatrix}$$

Parameter Estimation

$$\text{Sample mean } \mathbf{m}: m_i = \frac{\sum_{t=1}^{N} x_i^t}{N}, i = 1,...,d$$

$$\text{Covariance matrix } \mathbf{S}: s_{ij} = \frac{\sum_{t=1}^{N} \left(x_i^t - m_i\right)\left(x_j^t - m_j\right)}{N}$$

$$\text{Correlation matrix } \mathbf{R}: r_{ij} = \frac{s_{ij}}{s_i s_j}$$

Classification

- If $p(\mathbf{x} \mid C_i) \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$

$$p(\mathbf{x} \mid C_i) = \frac{1}{(2\pi)^{d/2} |\Sigma_i|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\right]$$

- Discriminant functions are

$$g_i(\mathbf{x}) = \log p(\mathbf{x} \mid C_i) + \log P(C_i)$$

$$= -\frac{d}{2}\log 2\pi - \frac{1}{2}\log |\Sigma_i| - \frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1} (\mathbf{x} - \mu_i) + \log P(C_i)$$

Estimating the mean and Variance,

$$\hat{P}(C_i) = \frac{\sum_t r_i^t}{N}$$

$$\mathbf{m}_i = \frac{\sum_t r_i^t \mathbf{x}^t}{\sum_t r_i^t}$$

$$\mathbf{S}_i = \frac{\sum_t r_i^t (\mathbf{x}^t - \mathbf{m}_i)(\mathbf{x}^t - \mathbf{m}_i)^T}{\sum_t r_i^t}$$

$$g_i(\mathbf{x}) = -\frac{1}{2}\log|\mathbf{S}_i| - \frac{1}{2}(\mathbf{x} - \mathbf{m}_i)^T \mathbf{S}_i^{-1}(\mathbf{x} - \mathbf{m}_i) + \log \hat{P}(C_i)$$

<u>Quadratic Discriminant:</u>

$$g_i(\mathbf{x}) = -\frac{1}{2}\log|\mathbf{S}_i| - \frac{1}{2}\left(\mathbf{x}^T \mathbf{S}_i^{-1}\mathbf{x} - 2\mathbf{x}^T \mathbf{S}_i^{-1}\mathbf{m}_i + \mathbf{m}_i^T \mathbf{S}_i^{-1}\mathbf{m}_i\right) + \log \hat{P}(C_i)$$

$$= \mathbf{x}^T \mathbf{W}_i \mathbf{x} + \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

where

$$\mathbf{W}_i = -\frac{1}{2}\mathbf{S}_i^{-1}$$

$$\mathbf{w}_i = \mathbf{S}_i^{-1}\mathbf{m}_i$$

$$w_{i0} = -\frac{1}{2}\mathbf{m}_i^T \mathbf{S}_i^{-1}\mathbf{m}_i - \frac{1}{2}\log|\mathbf{S}_i| + \log \hat{P}(C_i)$$

Tuning Complexity

Assumption	*Covariance matrix*	*No of parameters*
Shared, Hyperspheric	$\mathbf{S}_i=\mathbf{S}=s^2\mathbf{I}$	1
Shared, Axis-aligned	$\mathbf{S}_i=\mathbf{S}$, with $s_{ij}=0$	d
Shared, Hyperellipsoidal	$\mathbf{S}_i=\mathbf{S}$	$d(d+1)/2$
Different, Hyperellipsoidal	$\mathbf{S}_i$	$K\,d(d+1)/2$

- As we increase complexity (less restricted **S**), bias decreases and variance increases
- Assume simple models (allow some bias) to control variance (regularization)

Discrete Features

- Binary features: $p_{ij} \equiv p(x_j = 1 \mid C_i)$

  if x_j are independent (Naive Bayes')

$$p(\mathbf{x} \mid C_i) = \prod_{j=1}^{d} p_{ij}^{x_j} \left(1 - p_{ij}\right)^{(1-x_j)}$$

  the discriminant is linear

$$g_i(\mathbf{x}) = \log p(\mathbf{x} \mid C_i) + \log P(C_i)$$
$$= \sum_j \left[x_j \log p_{ij} + \left(1 - x_j\right) \log \left(1 - p_{ij}\right)\right] + \log P(C_i)$$

Estimated parameters $\hat{p}_{ij} = \dfrac{\sum_t x_j^t r_i^t}{\sum_t r_i^t}$

- Multinomial (1-of-n_j) features: $x_j \in \{v_1, v_2, ..., v_{n_j}\}$

$$p_{ijk} \equiv p(z_{jk} = 1 \mid C_i) = p(x_j = v_k \mid C_i)$$

if x_j are independent

$$p(\mathbf{x} \mid C_i) = \prod_{j=1}^{d} \prod_{k=1}^{n_j} p_{ijk}^{z_{jk}}$$

$$g_i(\mathbf{x}) = \sum_j \sum_k z_{jk} \log p_{ijk} + \log P(C_i)$$

$$\hat{p}_{ijk} = \frac{\sum_t z_{jk}^t r_i^t}{\sum_t r_i^t}$$

Dimensionality Reduction

Necessity:

1. Reduces time complexity: Less computation
2. Reduces space complexity: Less parameters
3. Saves the cost of observing the feature
4. Simpler models are more robust on small datasets
5. More interpretable; simpler explanation
6. Data visualization (structure, groups, outliers, etc) if plotted in 2or 3 dimensions.

Feature Selection and Extraction:

- Feature selection: Choosing $k<d$ important features, ignoring the remaining $d - k$

  Subset selection algorithms

- Feature extraction: Project the original x_i , $i=1,...,d$ dimensions to new $k<d$ dimensions, z_j , $j=1,...,k$

  Principal components analysis (PCA), linear discriminant analysis (LDA), factor analysis (FA)

Principal Component Analysis(PCA)

- Find a low-dimensional space such that when $\boldsymbol{x}$ is projected there, information loss is minimized.
- The projection of $\boldsymbol{x}$ on the direction of $\boldsymbol{w}$ is: $z = \boldsymbol{w}^T\boldsymbol{x}$
- Find $\boldsymbol{w}$ such that Var(z) is maximized

$$\begin{aligned}\text{Var}(z) &= \text{Var}(\boldsymbol{w}^T\boldsymbol{x}) = E[(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})^2] \\ &= E[(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})] \\ &= E[\boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{w}] \\ &= \boldsymbol{w}^T E[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T]\boldsymbol{w} = \boldsymbol{w}^T\boldsymbol{\Sigma}\,\boldsymbol{w}\end{aligned}$$

where $\text{Var}(\boldsymbol{x}) = E[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T] = \boldsymbol{\Sigma}$

- Maximize Var(z) subject to $\|\boldsymbol{w}\|=1$

$$\max_{\boldsymbol{w}_1} \boldsymbol{w}_1^T \boldsymbol{\Sigma} \boldsymbol{w}_1 - \alpha\left(\boldsymbol{w}_1^T \boldsymbol{w}_1 - 1\right)$$

$\boldsymbol{\Sigma w}_1 = \alpha \boldsymbol{w}_1$ that is, $\boldsymbol{w}_1$ is an eigenvector of $\boldsymbol{\Sigma}$

Choose the one with the largest eigenvalue for Var(z) to be max

- Second principal component: Max Var(z_2), s.t., $\|\boldsymbol{w}_2\|=1$ and orthogonal to $\boldsymbol{w}_1$

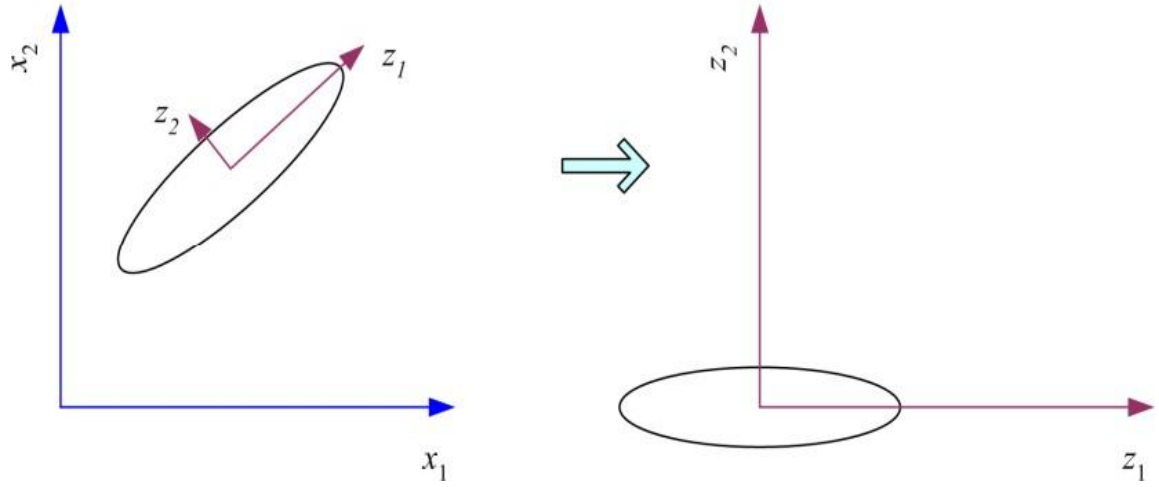
$$\max_{\boldsymbol{w}_2} \boldsymbol{w}_2^T \boldsymbol{\Sigma} \boldsymbol{w}_2 - \alpha\left(\boldsymbol{w}_2^T \boldsymbol{w}_2 - 1\right) - \beta\left(\boldsymbol{w}_2^T \boldsymbol{w}_1 - 0\right)$$

$\boldsymbol{\Sigma}\, \boldsymbol{w}_2 = \alpha\, \boldsymbol{w}_2$ that is, $\boldsymbol{w}_2$ is another eigenvector of $\boldsymbol{\Sigma}$ and so on.

$$\boldsymbol{z} = \mathbf{W}^T(\boldsymbol{x} - \boldsymbol{m})$$

where the columns of $\mathbf{W}$ are the eigenvectors of $\boldsymbol{\Sigma}$, and $\boldsymbol{m}$ is sample mean

Centers the data at the origin and rotates the axes



- Proportion of Variance (PoV) explained

$$\frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_k + \cdots + \lambda_d}$$

when λ_i are sorted in descending order

- Typically, stop at PoV>0.9
- Scree graph plots of PoV vs k, stop at "elbow"

Factor Analysis

- Find a small number of factors $\boldsymbol{z}$, which when combined generate $\boldsymbol{x}$:

$$x_i - \mu_i = v_{i1}z_1 + v_{i2}z_2 + \ldots + v_{ik}z_k + \varepsilon_i$$

where z_j, $j=1,\ldots,k$ are the latent factors with

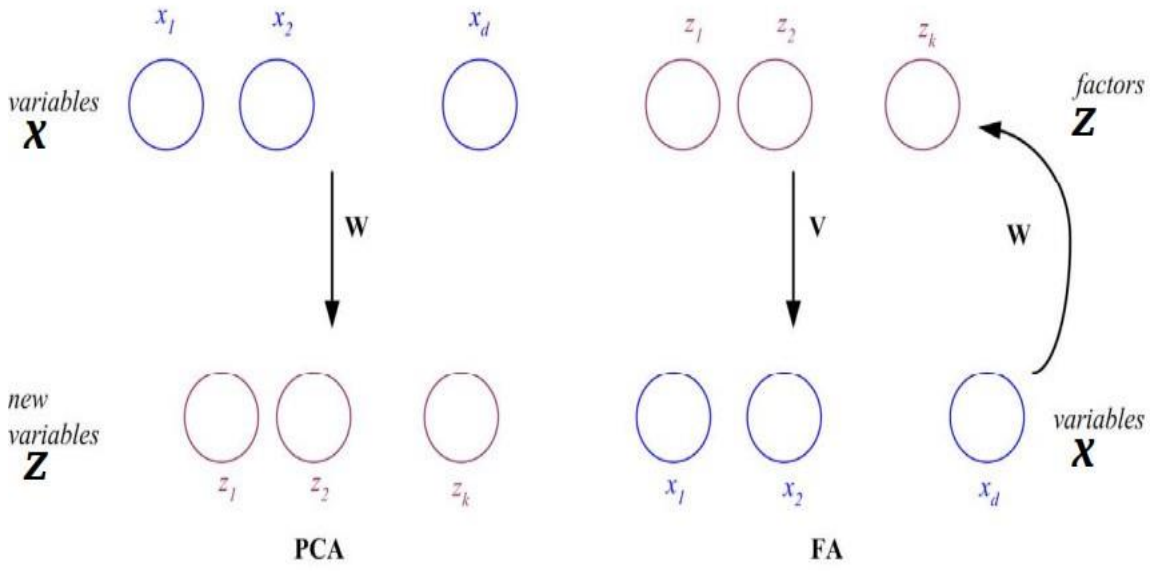
$$E[z_j]=0,\ Var(z_j)=1,\ Cov(z_i, z_j)=0,\ i \neq j,$$

ε_i are the noise sources

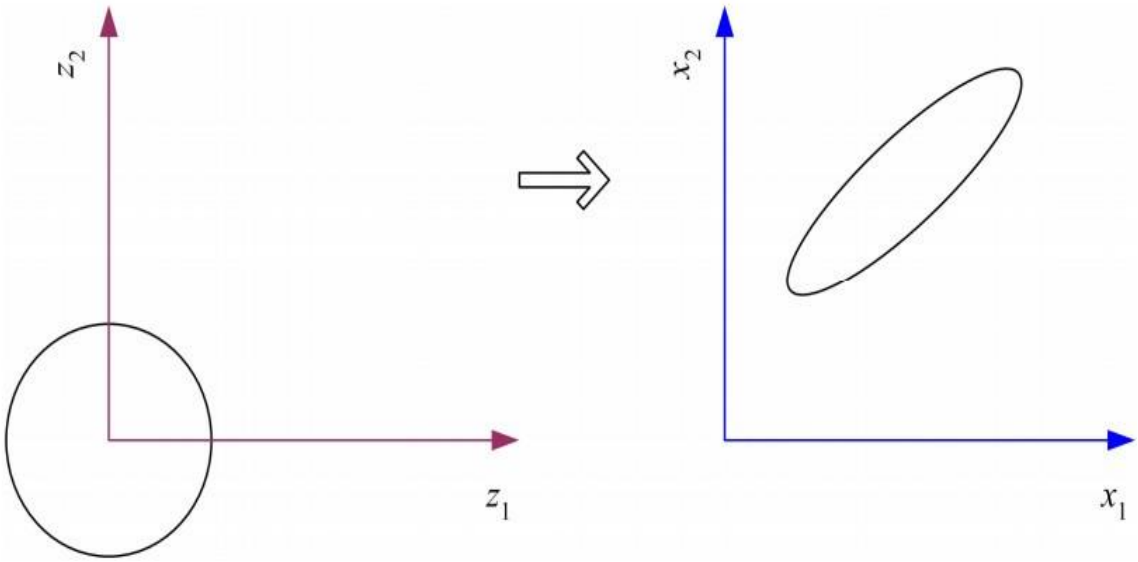
$$E[\varepsilon_i]=\psi_i,\ Cov(\varepsilon_i, \varepsilon_j)=0,\ i \neq j,\ Cov(\varepsilon_i, z_j)=0,$$

and v_{ij} are the factor loadings

- PCA From $\boldsymbol{x}$ to $\boldsymbol{z}$ $\boldsymbol{z} = \mathbf{W}^T(\boldsymbol{x} - \boldsymbol{\mu})$
- FA From $\boldsymbol{z}$ to $\boldsymbol{x}$ $\boldsymbol{x} - \boldsymbol{\mu} = \mathbf{V}\boldsymbol{z} + \boldsymbol{\varepsilon}$



- In FA, factors z_j are stretched, rotated and translated to generate $\boldsymbol{x}$



Multidimensional Scaling

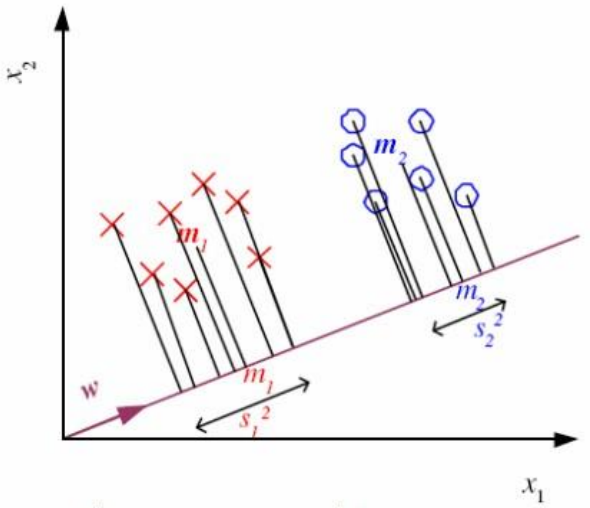
- Given pairwise distances between N points,
  d_{ij}, $i,j = 1,\ldots,N$
  place on a low-dim map s.t. distances are preserved.
- $\boldsymbol{z} = \boldsymbol{g}(\boldsymbol{x} \mid \theta)$ Find θ that min Sammon stress

$$E(\theta \mid \mathcal{X}) = \sum_{r,s} \frac{\left(\left\|\boldsymbol{z}^r - \boldsymbol{z}^s\right\| - \left\|\boldsymbol{x}^r - \boldsymbol{x}^s\right\|\right)^2}{\left\|\boldsymbol{x}^r - \boldsymbol{x}^s\right\|^2}$$

$$= \sum_{r,s} \frac{\left(\left\|\boldsymbol{g}(\boldsymbol{x}^r \mid \theta) - \boldsymbol{g}(\boldsymbol{x}^s \mid \theta)\right\| - \left\|\boldsymbol{x}^r - \boldsymbol{x}^s\right\|\right)^2}{\left\|\boldsymbol{x}^r - \boldsymbol{x}^s\right\|^2}$$

Linear Discriminant Analysis

- Find a low-dimensional space such that when $\boldsymbol{x}$ is projected, classes are well-separated.
- Find $\boldsymbol{w}$ that maximizes



$$J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

$$m_1 = \frac{\sum_t \boldsymbol{w}^T \boldsymbol{x}^t r^t}{\sum_t r^t} \qquad s_1^2 = \sum_t \left(\boldsymbol{w}^T \boldsymbol{x}^t - m_1\right)^2 r^t$$

- Between-class scatter:

$$(m_1 - m_2)^2 = \left(\mathbf{w}^T\mathbf{m}_1 - \mathbf{w}^T\mathbf{m}_2\right)^2$$
$$= \mathbf{w}^T(\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T\mathbf{w}$$
$$= \mathbf{w}^T\mathbf{S}_B\mathbf{w} \text{ where } \mathbf{S}_B = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T$$

- Within-class scatter:

$$s_1^2 = \sum_t \left(\mathbf{w}^T\mathbf{x}^t - m_1\right)^2 r^t$$
$$= \sum_t \mathbf{w}^T\left(\mathbf{x}^t - \mathbf{m}_1\right)\left(\mathbf{x}^t - \mathbf{m}_1\right)^T\mathbf{w}r^t = \mathbf{w}^T\mathbf{S}_1\mathbf{w}$$

$$\text{where } \mathbf{S}_1 = \sum_t \left(\mathbf{x}^t - \mathbf{m}_1\right)\left(\mathbf{x}^t - \mathbf{m}_1\right)^T r^t$$

$$s_1^2 + s_1^2 = \mathbf{w}^T\mathbf{S}_W\mathbf{w} \text{ where } \mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2$$

Fisher's Linear Discriminant:

- Find $\mathbf{w}$ that max

$$J(\mathbf{w}) = \frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T\mathbf{S}_W\mathbf{w}} = \frac{\left|\mathbf{w}^T(\mathbf{m}_1 - \mathbf{m}_2)\right|^2}{\mathbf{w}^T\mathbf{S}_W\mathbf{w}}$$

- LDA soln:

$$\mathbf{w} = c \cdot \mathbf{S}_W^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$$

- Parametric soln:

$$\mathbf{w} = \Sigma^{-1}(\mu_1 - \mu_2)$$
$$\text{when } p(\mathbf{x} \mid C_i) \sim \mathcal{N}(\mu_i, \Sigma)$$

For K>2 Classes,

- Within-class scatter:

$$\mathbf{S}_W = \sum_{i=1}^{K} \mathbf{S}_i \qquad \mathbf{S}_i = \sum_t r_i^t \left(\mathbf{x}^t - \mathbf{m}_i\right)\left(\mathbf{x}^t - \mathbf{m}_i\right)^T$$

- Between-class scatter:

$$\mathbf{S}_B = \sum_{i=1}^{K} N_i \left(\mathbf{m}_i - \mathbf{m}\right)\left(\mathbf{m}_i - \mathbf{m}\right)^T \qquad \mathbf{m} = \frac{1}{K}\sum_{i=1}^{K} \mathbf{m}_i$$

- Find $\mathbf{W}$ that max

$$J(\mathbf{W}) = \frac{\left|\mathbf{W}^T \mathbf{S}_B \mathbf{W}\right|}{\left|\mathbf{W}^T \mathbf{S}_W \mathbf{W}\right|}$$

The largest eigenvectors of $\mathbf{S}_W^{-1}\mathbf{S}_B$
Maximum rank of K-1

TEXT / REFERENCE BOOKS

1. Chris Albon : Machine Learning with Python Cookbook , O"Reilly Media, Inc.2018
2. Stephen Marsland, "Machine Learning – An Algorithmic Perspective", Second Edition, Chapman and Hall/CRC Machine Learning and Pattern Recognition Series, 2014
3. Tom M. Mitchell, Machine Learning, India Edition 2013, McGraw Hill Education
4. Machine Learning: The art and Science of algorithms that make sense of data, Peter Flach, Cambridge University Press, 2012
5. EthemAlpaydın, Introduction to machine learning, second edition, MIT press.
6. T. Hastie, R. Tibshirani and J. Friedman, "Elements of Statistical Learning", Springer Series , 2nd edition
7. Sebastian Raschka, "Python Machine Learning",Second Edition.Packt Publication

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT-III SUPERVISED LEARNING

UNIT III

UNIT III SUPERVISED LEARNING

Regression, Types of Regression model, Building a Regressor in Python, Types of ML Algorithm, Linear Regression, Multiple Linear Regression, Non-linear Regression, Model evaluation methods.

REGRESSION:

Statistical modeling to show the relationship between two variables with the linear Regression

Regression analysis is form of predictive modeling technique which investigates the relationship between a dependent and independent variable

What is the strength of relationship between sales and marketing spending?

What is the relationship between age and income?

Agricultural scientists often use linear regression to measure the effect of fertilizer and water on crop yields.

Data scientists for professional sports teams often use linear regression to measure the effect that different training regimens have on player performance

Example:

Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog.

For this identification, we can use the KNN algorithm, as it works on a similarity measure.

Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Clustering

- Cluster: A collection of data objects
  - similar (or related) to one another within the same group
  - dissimilar (or unrelated) to the objects in other groups
- Cluster analysis (or clustering, data segmentation, …)
  - Finding similarities between data according to the characteristics found in the data and grouping similar data objects into clusters
- Unsupervised learning: no predefined classes (i.e., learning by observations vs. learning by examples: supervised)
- A good clustering method will produce high quality clusters
  - high intra-class similarity: cohesive within clusters
  - low inter-class similarity: distinctive between clusters
- The quality of a clustering method depends on
  - the similarity measure used by the method
  - its implementation, and
  - its ability to discover some or all of the hidden patterns.

Major Clustering Approaches

- Partitioning approach:
  - Construct various partitions and then evaluate them by some criterion, e.g., minimizing the sum of square errors
  - Typical methods: k-means, k-medoids, CLARANS

- Hierarchical approach:
  - Create a hierarchical decomposition of the set of data (or objects) using some

criterion
- Typical methods: DIANA, AGNES, BIRCH, CAMELEON

Mixture densities

The *mixture density* is written as

$$p(\boldsymbol{x}) = \sum_{i=1}^{k} p(\boldsymbol{x}|\mathcal{G}_i)P(\mathcal{G}_i)$$

where $\mathcal{G}_i$ are the *mixture components*. They are also called *group* or *clusters*. $p(\boldsymbol{x}|\mathcal{G}_i)$ are the *component densities* and $P(\mathcal{G}_i)$ are the *mixture proportions*. The number of components, k, is a hyperparameter and should be specified beforehand. Given a sample and k, learning corresponds to estimating the component densities and proportions. When we assume that the component densities obey a parametric model, we need only estimate their parameters. If the component densities are multivariate Gaussian, we have $p(\boldsymbol{x}|\mathcal{G}_i) \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, and $\Phi = \{P(\mathcal{G}_i), \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}_{i=1}^{k}$ are the parameters that should be estimated from the iid sample $\mathcal{X} = \{\boldsymbol{x}^t\}_t$.

Parametric classification is a bona fide mixture model where groups, $\mathcal{G}_i$, correspond to classes, C_i, component densities $p(\boldsymbol{x}|\mathcal{G}_i)$ correspond to class densities $p(\boldsymbol{x}|C_i)$, and $P(\mathcal{G}_i)$ correspond to class priors, $P(C_i)$:

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

In this *supervised* case, we know how many groups there are and learning the parameters is trivial because we are given the labels, namely, which instance belongs to which class (component). We remember from chapter 5 that when we are given the sample $\mathcal{X} = \{\boldsymbol{x}^t, \boldsymbol{r}^t\}_{t=1}^{N}$, where $r_i^t = 1$ if $\boldsymbol{x}^t \in C_i$ and 0 otherwise, the parameters can be calculated using maximum likelihood. When each class is Gaussian distributed, we have a Gaussian mixture, and the parameters are estimated as

$$\begin{aligned}
\hat{P}(C_i) &= \frac{\sum_t r_i^t}{N} \\
\boldsymbol{m}_i &= \frac{\sum_t r_i^t \boldsymbol{x}^t}{\sum_t r_i^t} \\
\mathbf{S}_i &= \frac{\sum_t r_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T}{\sum_t r_i^t}
\end{aligned}$$

The difference in this chapter is that the sample is $\mathcal{X} = \{\boldsymbol{x}^t\}_t$: We have an *unsupervised learning* problem. We are given only $\boldsymbol{x}^t$ and not the labels $\boldsymbol{r}^t$, that is, we do not know which $\boldsymbol{x}^t$ comes from which component. So we should estimate both: First, we should estimate the labels, r_i^t, the component that a given instance belongs to; and, second, once we estimate the labels, we should estimate the parameters of the components given the set of instances belonging to them. We are first going to discuss a simple algorithm, *k*-means clustering, for this purpose and later on show that it is a special case of the *Expectation-Maximization* algorithm.

Partitioning method

- Partitioning a database *D* of *n* objects into a set of *k* clusters, such that the sum of squared distances is minimized (where c_i is the centroid or medoid of cluster C_i)
- Given *k*, find a partition of *k clusters* that optimizes the chosen partitioning criterion

The K-Means Clustering Method

- Given *k*, the *k-means* algorithm is implemented in four steps:
  - Partition objects into *k* nonempty subsets
  - Compute seed points as the centroids of the clusters of the current partitioning (the centroid is the center, i.e., *mean point*, of the cluster)
  - Assign each object to the cluster with the nearest seed point
  - Go back to Step 2, stop when the assignment does not change

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

k-means algorithm.

Example:

K-Means clustering:-

1) Decide the number & frame the clusters.
2) Find Mean of clusters.
3) Find distance b/w Mean & distance for all points.

Eg:- $x_1 = \{1, 0\}$, $x_2 = \{0, 1\}$, $x_3 = \{2, 1\}$, $x_4 = \{2, 3\}$.

Assume, we take two clusters, @

$C_1 = \{x_1, x_3\}$ & $C_2 = \{x_2, x_4\}$

(a) Apply 1 iteration of K-means partional clustering algorithm

(b) What is change in total square error.

(c) Apply 2nd iteration of K-means algorithm.

step 1:-

Mean,

$$M_k = \frac{1}{n_k} \sum_{i=1}^{k} x_{ik}$$

C_1:-

$x_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

$x_2 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

$\Rightarrow M_1 = \left\{ \frac{1+2}{2}, \frac{0+1}{2} \right\}$

$= \{1.5, 0.5\}$.

$M_2 = \left\{ \frac{0+2}{2}, \frac{1+3}{2} \right\}$

$= \{1.5, 2\}$.

C_2 :- $x_3 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$ $x_4 = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$

Step 2 :-

Error Calculation.

$$e_k^2 = \sum_{i=1}^{k} (x_{ik} - M_{ik})^2$$

$M_1 = (1.5, 0.5)$ | $M_2 = (1.5, 2)$
$x_1 = (1, 0)$ | $x_3 = (0, 2)$
$x_2 = (2, 1)$ | $x_4 = (2, 3)$

$C_1 \Rightarrow e_1^2 = \left[(1-1.5)^2 + (0-0.5)^2 + (2-1.5)^2 + (1-0.5)^2 \right]$

$= 1.$

$C_2 \Rightarrow e_2^2 = \left[(0-1.5)^2 + (1-2)^2 + (3-1.5)^2 + (2-2)^2 \right]$

$= 6.5$

Step 3 :-

Total square error.

$$\Sigma_k^2 = \sum_{k=1}^{k} e_k^2$$

$\Rightarrow e_1^2 + e_2^2$

$1 + 6.5 = 7.5$

Step 4 :-

Distance calculation.

k-means follow euclidean's distance measure,

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

X_1 (1, 0):-

$d(X_1, M_1) = \sqrt{(1-1.5)^2 + (0-0.5)^2} = 0.707$

$d(X_1, M_2) = \sqrt{(1-1.5)^2 + (0-2)^2} = 2.062$

X_2 (0, 1):-

$d(X_2, M_1) = \sqrt{(0-1.5)^2 + (1-0.5)^2} = 1.581$

$d(X_2, M_2) = \sqrt{(0-1.5)^2 + (1-2)^2} = 1.803$

X_3 (2, 1):-

$d(X_3, M_1) = \sqrt{(2-1.5)^2 + (1-0.5)^2} = 0.707$

$d(X_3, M_2) = \sqrt{(2-1.5)^2 + (1-2)^2} = 1.118$

X_4 (3, 3):-

$d(X_4, M_1) = \sqrt{(3-1.5)^2 + (3-0.5)^2} = 2.915$

$d(X_4, M_2) = \sqrt{(3-1.5)^2 + (3-2)^2} = 1.8027$

	M_1	M_2
X_1	0.707	2.062
X_2	1.581	1.803
X_3	0.707	1.118
X_4	2.915	1.8027
	C_1	C_2

1) check the smallest element row wise

2) Divide the columns C_1 & C_2.

$\Rightarrow C_1 = (X_1, X_3, X_2)$

$C_2 = (X_4)$.

(c)

Mean,

$$M_i = \frac{1}{n_i} \sum_{i=1}^{r} x_{ij}$$

C_1:-

$x_1 = (1, 0)$

$x_2 = (0, 1)$ $\Rightarrow$ $M_1 = \left\{ \frac{1+0+0}{3}, \frac{0+1+1}{3} \right\}$

$x_3 = (2, 1)$ $= (1, 0.66)$

C_2:-

$x_4 = (3, 3)$ $\Rightarrow$ $M_2 = \{3, 3\}$.

Error Calculation,

$C_1 \Rightarrow e_1^2 \Rightarrow [(1-1)^2 + (0-0.66)^2 + (0-1)^2 + (1-0.66)^2 + (2-1)^2 + (1-0.66)^2]$

$= 0.4356 + 1 + 0.1156 + 1 + 0.1156 = 1.6668.$

$C_2 \Rightarrow e_2^2 \Rightarrow [(3-3)^2 + (3-3)^2] = 0.$

Total square error

$\Rightarrow e_1^2 + e_2^2 = 1.6668$

Distance Calculation,

$X_1 (1, 0)$

$d(x_1, M_1) = \sqrt{(1-1)^2 + (0-0.66)^2} = \sqrt{0 + 0.4356} = 0.66$

$d(x_2, M_2) = \sqrt{(1-1)^2 + (0-3)^2} = \sqrt{4+9} = \sqrt{13}$

$= 3.605$

$X_2\ (0,1)$

$$d(x_2, M_1) = \sqrt{(0-1)^2 + (0.666)(1-0.66)^2} = \sqrt{1+0.11} = 1.057$$

$$d(x_2, M_2) = \sqrt{(0-2)^2 + (1-3)^2} = \sqrt{9+4} = \sqrt{13} = 2.61$$

$X_3\ (2,1)$

$$d(x_3, M_1) = \sqrt{(2-1)^2 + (1-0.666)^2} = 1.056$$

$$d(x_3, M_2) = \sqrt{(2-3)^2 + (1-2)^2} = \sqrt{5} = 2.236$$

$X_4\ (3,3)$

$$d(x_4, M_1) = \sqrt{(3-1)^2 + (3-0.66)^2} = 9.4756.$$

$$d(x_4, M_2) = \sqrt{(3-3)^2 + (3-2)^2} = 0.$$

	M_1	M_2
X_1	0.66 ✓	3.605
X_2	1.056 ✓	2.605
X_3	1.056 ✓	2.236
X_4	9.4756	0 ✓
	C_1	C_2

$$\Rightarrow \quad C_1 = (x_1, x_3, x_2)$$

$$C_2 = (x_4).$$

The iteration stops here, as we get the same set of clusters as the previous step.

Agglomerative Clustering:

Start with N groups each with one instance and merge two closest groups at each iteration

Distance between two groups Gi and Gj:

Single-link clustering:

$$d(G_i, G_j) = \min_{\mathbf{x}^r \in G_i, \mathbf{x}^s \in G_j} d(\mathbf{x}^r, \mathbf{x}^s)$$

Complete-link clustering:

$$d(G_i, G_j) = \max_{\mathbf{x}^r \in G_i, \mathbf{x}^s \in G_j} d(\mathbf{x}^r, \mathbf{x}^s)$$

Average-link clustering, centroid

Dendrogram: Decompose data objects into a several levels of nested partitioning(tree of clusters), called a dendrogram

A clustering of the data objects is obtained by cutting the dendrogram at thedesired level, then each connected component forms a cluster

Example of Single Linkage Clustering:

1. Given a datset, first find the distance matrix using Euclidean distance measure.
2. After finding the distance matrix, find the smallest element in the distance matix.
3. Merge these two points to form a cluster.
4. Next find the minium distance between the new cluster obtained with all other points.
5. Now frame the distance matrix and find the smallest value in the obtained distance matrix and then merge these points to form a cluster. The process repeats until all points are grouped into clusters.
6. Finally draw the dendogram.

Example:

	X	Y
P1	0.40	0.53
P2	0.22	0.38
P3	0.35	0.32
P4	0.26	0.19
P5	0.08	0.41
P6	0.45	0.30

Find clusters using single linkage technique.

Solution:

Step 1:-

Euclidean distance.

$$d(P_1,P_2) = \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2}.$$

$$= \sqrt{0.0549} = 0.234.$$

$d(P_1,P_3) = \sqrt{0.0466} = 0.215$

$d(P_2,P_3) = \sqrt{0.0205} = 0.1431.$

$d(P_1,P_4) = \sqrt{0.1352} = 0.3676$

$d(P_2,P_4) = \sqrt{0.0377} = 0.1941.$

$d(P_3,P_4) = \sqrt{0.025} = 0.1581.$

$d(P_1,P_5) = 0.34$

$d(P_2,P_5) = 0.14$

$d(P_3,P_5) = 0.28$

$d(P_4,P_5) = 0.27$

$P(P_1,P_6) = 0.23$

$d(P_2,P_6) = 0.25$

$d(P_3,P_6) = 011$

$d(P_4,P_6) = 0.22$

$d(P_5,P_6) = 0.37$

2) Get the distance matrix.

	P1	P2	P3	P4	P5	P6
P1	0					
P2	0.234	0				
P3	0.22	0.15	0			
P4	0.37	0.20	0.15	0		
P5	0.34	0.14	0.28	0.29	0	
P6	0.23	0.25	**0.11**	0.22	0.39	0

↳ smallest value

3) Find minimum element from the dist. matrix.

A) $P_3 P_6$:-

- Recalculate the dist matrix.
- To update the dist. matrix.,

$$\min(dist(P_3, P_6), P_1)$$
$$= \min(dist(P_3, P_1), (P_6, P_1))$$
$$= \min(0.22, 0.23) = 0.22$$

$$\min(dist(P_3, P_6), P_2)$$
$$= \min(dist(P_3, P_2), (P_6, P_2))$$
$$= \min(0.15, 0.25) = 0.15$$

$$\min(dist(P_3, P_6), P_4)$$
$$= \min(dist(P_3, P_4), (P_6, P_4))$$
$$= \min(0.15, 0.22) = 0.15$$

$$\min(dist(P_3, P_6), P_5)$$
$$= \min(dist(P_3, P_5), (P_6, P_5))$$
$$= \min(0.28, 0.39) = 0.28$$

Now, the new clusters formed are (P_3, P_6).

Updated values for clusters w.r.t $(P_3\ P_6)$ are :-

	P_1	P_2	P_3, P_6	P_4	P_5
P_1	0				
P_2	0.23	0			
P_3, P_6	0.22	0.15	0		
P_4	0.37	0.20	0.15	0	
P_5	0.34	**0.14** ↳ smallest element.	0.28	0.29	0

New cluster - $(P_2\ P_5)$:-

Update the dist. matrix.

$\min(dist(P_2, P_5), P_1)$

$= \min(dist[(P_2, P_1), (P_5, P_1)]$

$= \min(0.23, 0.34) = 0.23$

$\min(dist(P_2, P_5), (P_3, P_6))$

$= \min(dist[(P_2, (P_3 P_6)), (P_5, (P_3 P_6))]$

$= \min(0.15, 0.28) = 0.15$

$\min(dist(P_2, P_5), P_4)$

$= \min(dist[(P_2, P_4), (P_5, P_4)]$

$= \min(0.20, 0.29) = 0.20.$

Updated dist. matrix :-

	P_1	P_2P_5	P_3P_6	P_4
P_1	0			
P_2P_5	0.23	0		
P_3P_6	0.22	[0.15]	0	
P_4	0.37	0.20	0.18	0

New cluster - $[(P_3P_6)(P_2P_5)]$:-

Update the dist matrix

$\min\left(\text{dist}\,[(P_2P_5)(P_3P_6)], P_1\right)$

$= \min\left(\text{dist}\,[(P_2P_5), P_1][(P_3P_6), P_1]\right)$

$= \min(0.23, 0.22) = 0.22.$

$\min(\text{dist}\,[(P_2P_5)(P_3P_6)], P_4)$

$= \min(\text{dist}\,[(P_2P_5), P_4]$
$[(P_3P_6), P_4]$

$= \min(0.20, 0.18)$

$= 0.18$

Updated dist. matrix is:-

	P_1	P_2 P_5 P_3 P_6	P_4
P_1	0		
P_2 P_5 P_3 P_6	0.22.	0	
P_4	0.37	0.15	0

New cluster - $[P_4\ P_2\ P_5\ P_3\ P_6]$:-

Update the dist. matrix is-

$$\min(\text{dist}[P_2 P_5 P_3 P_6\ P_4], P_1)$$

$$= \min(\text{dist}[P_2 P_5 P_3 P_6, P_1], [P_4, P_1])$$

$$= \min(0.22, 0.37) = 0.22.$$

⇒ Updated dist. matrix:-

	P_1	$P_2\ P_5\ P_3\ P_6\ P_4$
P_1	0	
$P_2\ P_5\ P_3\ P_6\ P_4$	0.22.	0

Dendogram:-

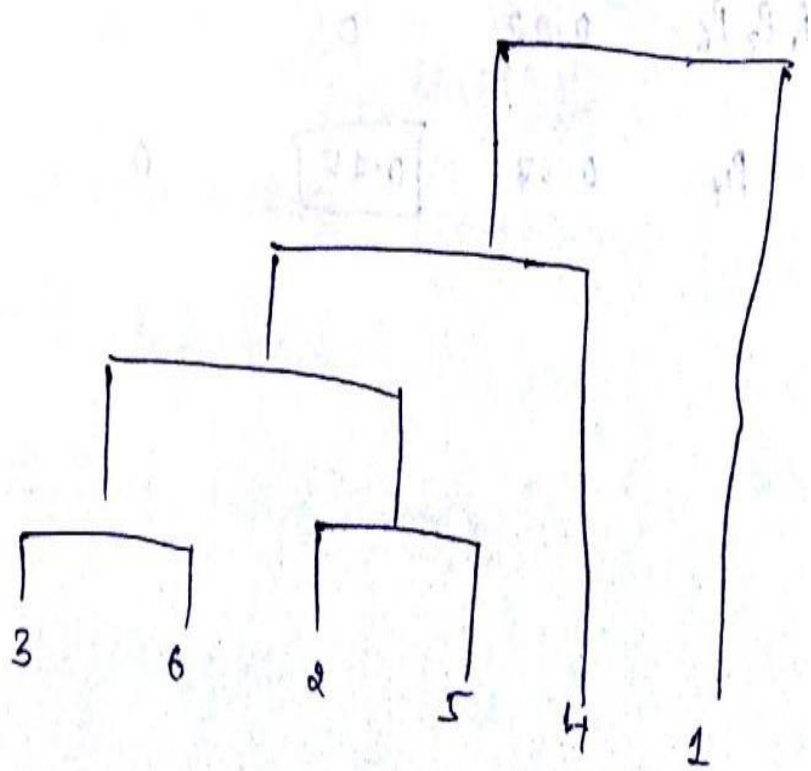
$P_3 P_6$

$P_2 P_5$

$(P_3 P_6)(P_2 P_5)$

$(P_3 P_6)(P_2 P_5) P_4$

$(P_3 P_6)(P_2 P_5) P_4 P_1$



Nonparametric Methods

- Parametric (single global model), semiparametric (small number of local models)
- Nonparametric: Similar inputs have similar outputs
- Functions (pdf, discriminant, regression) change smoothly
- Keep the training data;"let the data speak for itself"
- Given x, find a small number of closest training instances and interpolate from these
- Aka lazy/memory-based/case-based/instance-based learning

Density Estimation

- Given the training set X={xt}t drawn iid from p(x)
- Divide data into bins of size h
- Histogram:

$$\hat{p}(x) = \frac{\#\{x^t \text{ in the same bin as } x\}}{Nh}$$

- Naive estimator:

$$\hat{p}(x) = \frac{\#\{x - h < x^t \le x + h\}}{2Nh}$$

or

$$\hat{p}(x) = \frac{1}{Nh}\sum_{t=1}^{N} w\left(\frac{x - x^t}{h}\right) \quad w(u) = \begin{cases} 1/2 & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

with the weight function defined as,,

$$w(u) = \begin{cases} 1 & \text{if } |u| < 1/2 \\ 0 & \text{otherwise} \end{cases}$$

This is as if each x^t has a symmetric region of influence of size h around it and contributes 1 for an x falling in its region. Then the nonparametric estimate is just the sum of influences of x^t whose regions include x. Because this region of influence is "hard" (0 or 1), the estimate is not a continuous function and has jumps at $x^t \pm h/2$.

Kernel Estimator:

- Kernel function, e.g., Gaussian kernel:

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{u^2}{2}\right]$$

- Kernel estimator (Parzen windows)

$$\hat{p}(x) = \frac{1}{Nh} \sum_{t=1}^{N} K\left(\frac{x - x^t}{h} \right)$$

The kernel function $K(\cdot)$ determines the shape of the influences and the window width h determines the width. Just like the naive estimate is the sum of "boxes," the kernel estimate is the sum of "bumps." All the x^t have an effect on the estimate at x, and this effect decreases smoothly as $|x - x^t|$ increases.

To simplify calculation, $K(\cdot)$ can be taken to be 0 if $|x - x^t| > 3h$. There exist other kernels easier to compute that can be used, as long as $K(u)$ is maximum for $u = 0$ and decreasing symmetrically as $|u|$ increases.

k-Nearest Neighbor Estimator:

- Instead of fixing bin width h and counting the number of instances, fix the instances(neighbors) k and check bin width,

$$\hat{p}(x) = \frac{k}{2Nd_k(x)}$$

$d_k(x)$, distance to kth closest instance to x.

The nearest neighbor class of estimators adapts the amount of smoothing to the local density of data. The degree of smoothing is controlled by k, the number of neighbors taken into account, which is much smaller than N, the sample size. Let us define a distance between a and b, for example, |a − b|, and for each x, we define,

$$d_1(x) \leq d_2(x) \leq \cdots \leq d_N(x)$$

to be the distances arranged in ascending order, from x to the points in the sample: $d_1(x)$ is the distance to the nearest sample, $d_2(x)$ is the distance to the next nearest, and so on. If x^t are the data points, then we define $d_1(x) = \min_t |x - x^t|$, and if i is the index of the closest sample, namely, $i = \arg\min_t |x - x^t|$, then $d_2(x) = \min_{j \neq i} |x - x^j|$, and so forth.

Generalization of multivariate data

- Kernel density estimator

$$\hat{p}(\mathbf{x}) = \frac{1}{Nh^d} \sum_{t=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{x}^t}{h}\right)$$

- Multivariate Gaussian kernel
  - Spheric $$K(\mathbf{u}) = \left(\frac{1}{\sqrt{2\pi}}\right)^d \exp\left[-\frac{\|\mathbf{u}\|^2}{2}\right]$$
  - Ellipsoid $$K(\mathbf{u}) = \frac{1}{(2\pi)^{d/2}|\mathbf{S}|^{1/2}} \exp\left[-\frac{1}{2}\mathbf{u}^T\mathbf{S}^{-1}\mathbf{u}\right]$$

where S is the sample covariance matrix. This corresponds to using Mahalanobis distance instead of the Euclidean distance. It is also possible to have the distance metric local where S is calculated from instances in the vicinity of x, for example, some k closest instances.
Note that S calculated locally may be singular and PCA (or LDA, in thecase of classification) may be needed.

Hamming distance:

When the inputs are discrete, we can use Hamming distance, which counts the number of nonmatching attributes.

$$HD(\mathbf{x}, \mathbf{x}^t) = \sum_{j=1}^{d} 1(x_j \neq x_j^t)$$

where

$$1(x_j \neq x_j^t) = \begin{cases} 1 & \text{if } x_j \neq x_j^t \\ 0 & \text{otherwise} \end{cases}$$

$HD(\mathbf{x}, \mathbf{x}^t)$ is then used in place of $\|\mathbf{x} - \mathbf{x}^t\|$ or $(\mathbf{x} - \mathbf{x}^t)^T\mathbf{S}^{-1}(\mathbf{x} - \mathbf{x}^t)$ for kernel estimation or for finding the k closest neighbors.

Nonparametric Classification

- Estimate $p(\mathbf{x}|C_i)$ and use Bayes' rule
- Kernel estimator

$$\hat{p}(\mathbf{x}\mid C_i)=\frac{1}{N_i h^d}\sum_{t=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{x}^t}{h}\right) r_i^t \qquad \hat{P}(C_i)=\frac{N_i}{N}$$

$$g_i(\mathbf{x})=\hat{p}(\mathbf{x}\mid C_i)\hat{P}(C_i)=\frac{1}{Nh^d}\sum_{t=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{x}^t}{h}\right) r_i^t$$

- k-NN estimator

$$\hat{p}(\mathbf{x}\mid C_i)=\frac{k_i}{N_i V^k(\mathbf{x})} \qquad \hat{P}(C_i\mid\mathbf{x})=\frac{\hat{p}(\mathbf{x}\mid C_i)\hat{P}(C_i)}{\hat{p}(\mathbf{x})}=\frac{k_i}{k}$$

The k-nn classifier assigns the input to the class having most examples among the k neighbors of the input. All neighbors have equal vote, and the class having the maximum number of voters among the k neighbors is chosen. Ties are broken arbitrarily or a weighted vote is taken. K is generally taken to be an odd number to minimize ties: confusion is generally between two neighboring classes. Again, the use of Euclidean distance corresponds to assuming uncorrelated inputs with equal variances, and when this is not the case a suitable discriminant metric should be used. One example is discriminant adaptive nearest neighbor where the optimal distance to separate classes is estimated locally.

Nonparametric Regression

- Aka smoothing models
- Regressogram

$$\hat{g}(x)=\frac{\sum_{t=1}^{N} b(x,x^t)\,r^t}{\sum_{t=1}^{N} b(x,x^t)}$$

where

$$b(x,x^t)=\begin{cases}1 & \text{if } x^t \text{ is in the same bin with } x\\ 0 & \text{otherwise}\end{cases}$$

Running Mean/Kernel Smoother:

- Running mean smoother

$$\hat{g}(x)=\frac{\sum_{t=1}^{N} w\left(\frac{x-x^t}{h}\right) r^t}{\sum_{t=1}^{N} w\left(\frac{x-x^t}{h}\right)}$$

where

$$w(u)=\begin{cases}1 & \text{if } |u|<1\\ 0 & \text{otherwise}\end{cases}$$

- Kernel smoother

$$\hat{g}(x)=\frac{\sum_{t=1}^{N} K\left(\frac{x-x^t}{h}\right) r^t}{\sum_{t=1}^{N} K\left(\frac{x-x^t}{h}\right)}$$

where $K(\,)$ is Gaussian

- Additive models (Hastie and Tibshirani, 1990)

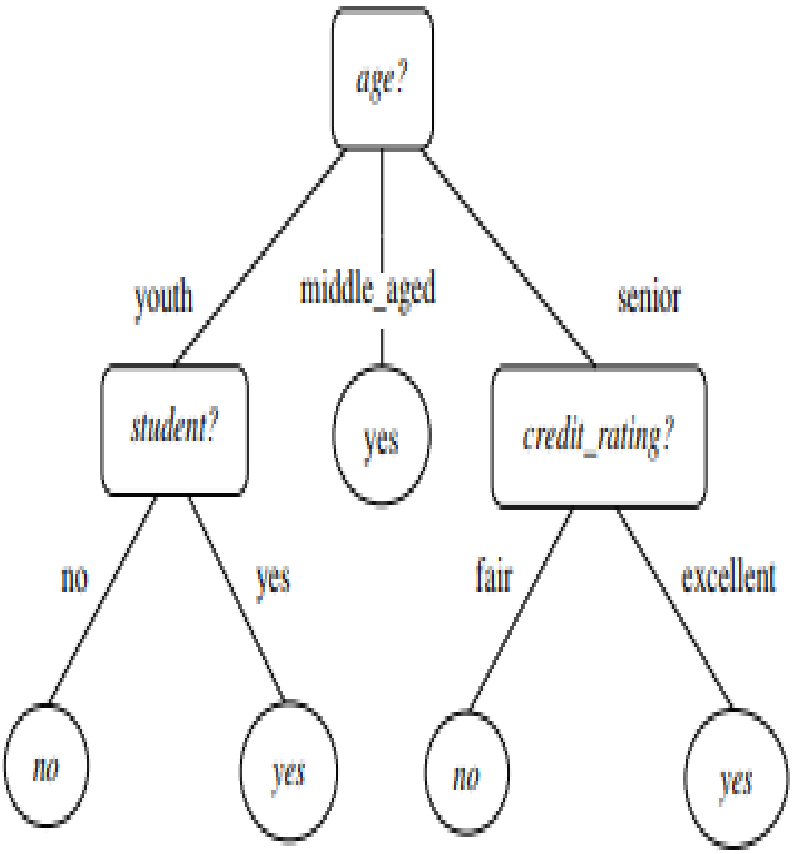
- Running line smoother

Instead of taking an average and giving a constant fit at a point, we can take into account one more term in the Taylor expansion and calculate a linear fit. In the running line smoother, we can use the data points in the neighborhood, as defined by h or k, and fit a local regression line. In the locally weighted running line smoother, known as loess, instead of a hard definition of neighborhoods, we use kernel weighting such that distant points have less effect on error.

Decision Trees

Decision tree induction is the learning of decision trees from class-labeled training tuples. A **decision tree** is a flowchart-like tree structure, where each **internal node** (non-leaf node) denotes a test on an attribute, each **branch** represents an outcome of the test, and each **leaf node** (or *terminal node*) holds a class label. The topmost node in a tree is the **root** node. A typical decision tree is shown in Figure 8.2. It represents the concept *buys_computer*, that is, it predicts whether a customer at *AllElectronics* is

likely to purchase a computer. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals. Some decision tree algorithms produce only *binary* trees (where each internal node branches to exactly two other nodes), whereas others can produce nonbinary trees.



A decision tree for the concept *buys_computer*, indicating whether an *AllElectronics* customer is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer* = *yes* or *buys_computer* = *no*).

Algorithm:

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, *D*.

Input:

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.

Method:

(1) create a node *N*;
(2) **if** tuples in *D* are all of the same class, *C*, **then**
(3) return *N* as a leaf node labeled with the class *C*;
(4) **if** *attribute_list* is empty **then**
(5) return *N* as a leaf node labeled with the majority class in *D*; // majority voting
(6) apply **Attribute_selection_method**(*D*, *attribute_list*) to **find** the "best" *splitting_criterion*;
(7) label node *N* with *splitting_criterion*;
(8) **if** *splitting_attribute* is discrete-valued **and**
multiway splits allowed **then** // not restricted to binary trees
(9) *attribute_list* ← *attribute_list* − *splitting_attribute*; // remove *splitting_attribute*
(10) **for each** outcome *j* of *splitting_criterion*
// partition the tuples and grow subtrees for each partition
(11) let D_j be the set of data tuples in *D* satisfying outcome *j*; // a partition
(12) **if** D_j is empty **then**
(13) attach a leaf labeled with the majority class in *D* to node *N*;
(14) **else** attach the node returned by **Generate_decision_tree**(D_j, *attribute_list*) to node *N*;
endfor
(15) return *N*;

Attribute Selection Measure:

Select the attribute with the highest information gain

Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$

Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

Information needed (after using A to split D into v partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Example:

Class-Labeled Training Tuples from the *AllElectronics* Customer Database

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Induction of a decision tree using information gain. Table 8.1 presents a training set, D, of class-labeled tuples randomly selected from the *AllElectronics* customer database. (The data are adapted from Quinlan [Qui86]. In this example, each attribute is discrete-valued. Continuous-valued attributes have been generalized.) The class label attribute, *buys_computer*, has two distinct values (namely, {*yes*, *no*}); therefore, there are two distinct classes (i.e., $m = 2$). Let class C_1 correspond to *yes* and class C_2 correspond to *no*. There are nine tuples of class *yes* and five tuples of class *no*. A (root) node N is created for the tuples in D. To find the splitting criterion for these tuples, we must compute the information gain of each attribute. We compute the expected information needed to classify a tuple in D:

$$Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$$

Next, we need to compute the expected information requirement for each attribute. Let's start with the attribute *age*. We need to look at the distribution of *yes* and *no* tuples for each category of *age*. For the *age* category "youth," there are two *yes* tuples and three *no* tuples. For the category "middle_aged," there are four *yes* tuples and zero *no* tuples. For the category "senior," there are three *yes* tuples and two *no* tuples. Using Eq. (8.2), the expected information needed to classify a tuple in D if the tuples are partitioned according to *age* is

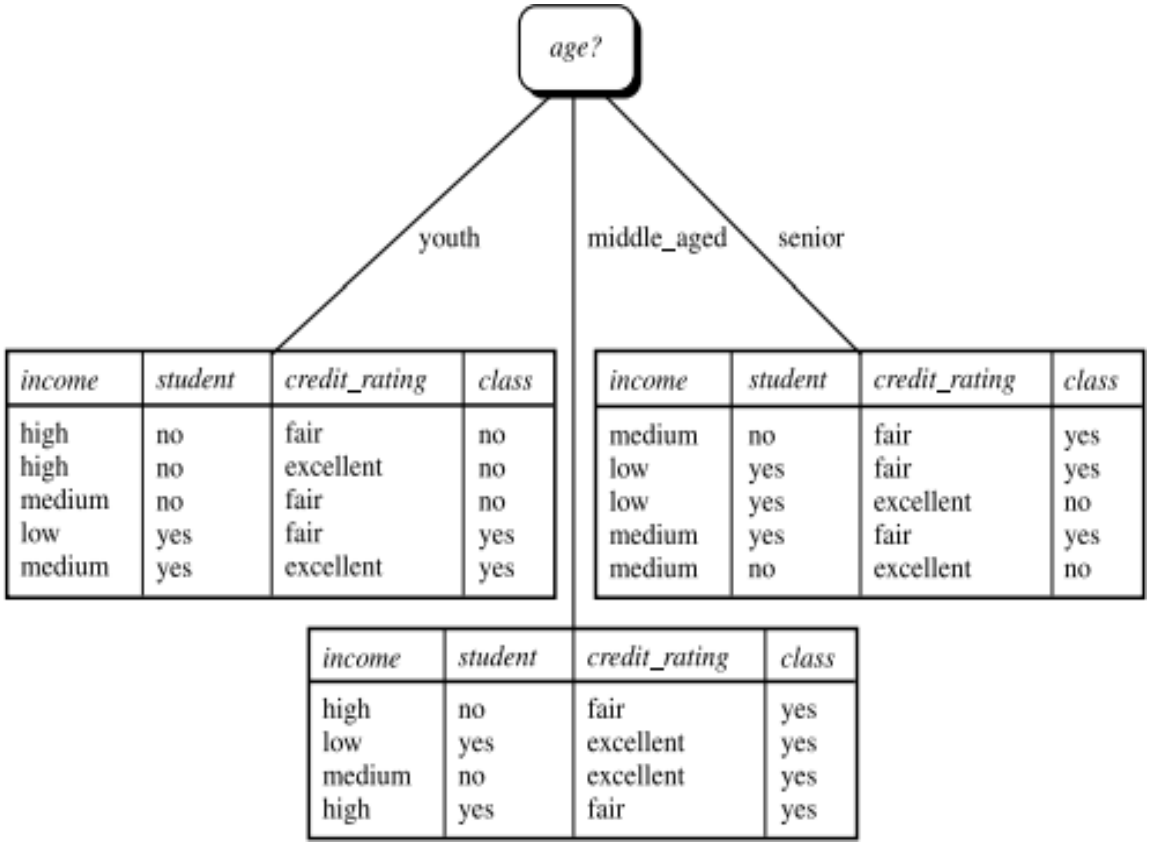
$$\begin{aligned} Info_{age}(D) = {} & \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) \\ & + \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4}\right) \\ & + \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right) \\ = {} & 0.694 \end{aligned}$$

Hence, the gain in information from such a partitioning would be

$$Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246$$

Similarly, we can compute $Gain(income) = 0.029$
$Gain(student) = 0.151$
and $Gain(credit\_rating) = 0.048$

Because *age* has the highest information gain among the attributes, it is selected as the splitting attribute. Node N is labeled with *age*, and branches are grown for each of the attribute's values. The tuples are then partitioned accordingly, as shown in Figure 8.5. Notice that the tuples falling into the partition for $age = middle\_aged$ all belong to the same class. Because they all belong to class "*yes*," a leaf should therefore be created at the end of this branch and labeled "*yes*."

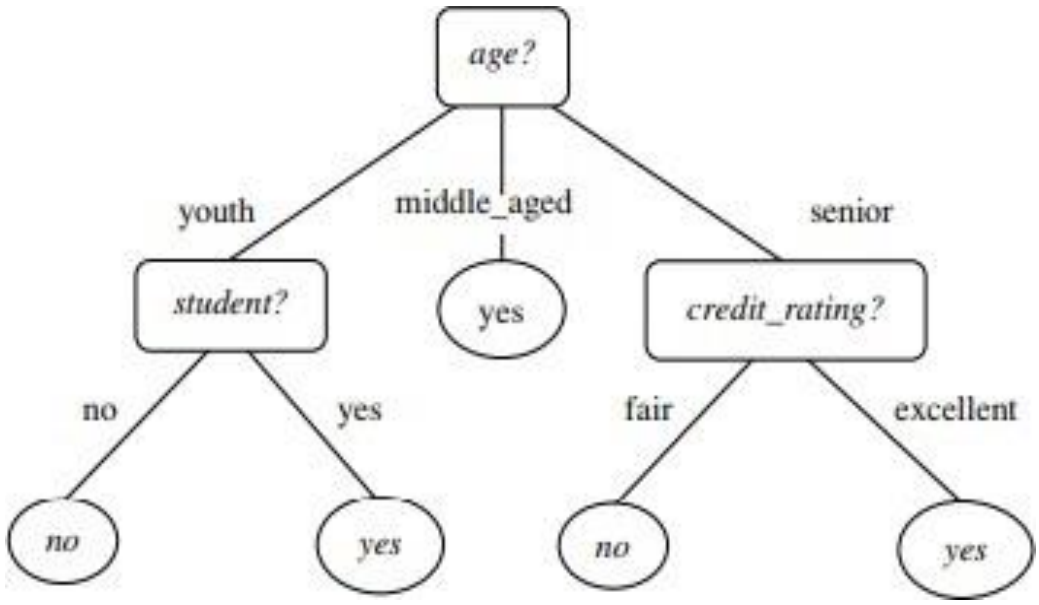


income	student	credit_rating	class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	student	credit_rating	class
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

income	student	credit_rating	class
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

After further iterations, the final decision tree would be,



In a *univariate tree*, in each internal node, the test uses only one of the input dimensions. If the used input dimension, x_j, is discrete, taking one of n possible values, the decision node checks the value of x_j and takes the corresponding branch, implementing an n-way split. For example, if an attribute is color $\in$ {red, blue, green}, then a node on that attribute has three branches, each one corresponding to one of the three possible values of the attribute.

A decision node has discrete branches and a numeric input should be discretized. If x_j is numeric (ordered), the test is a comparison

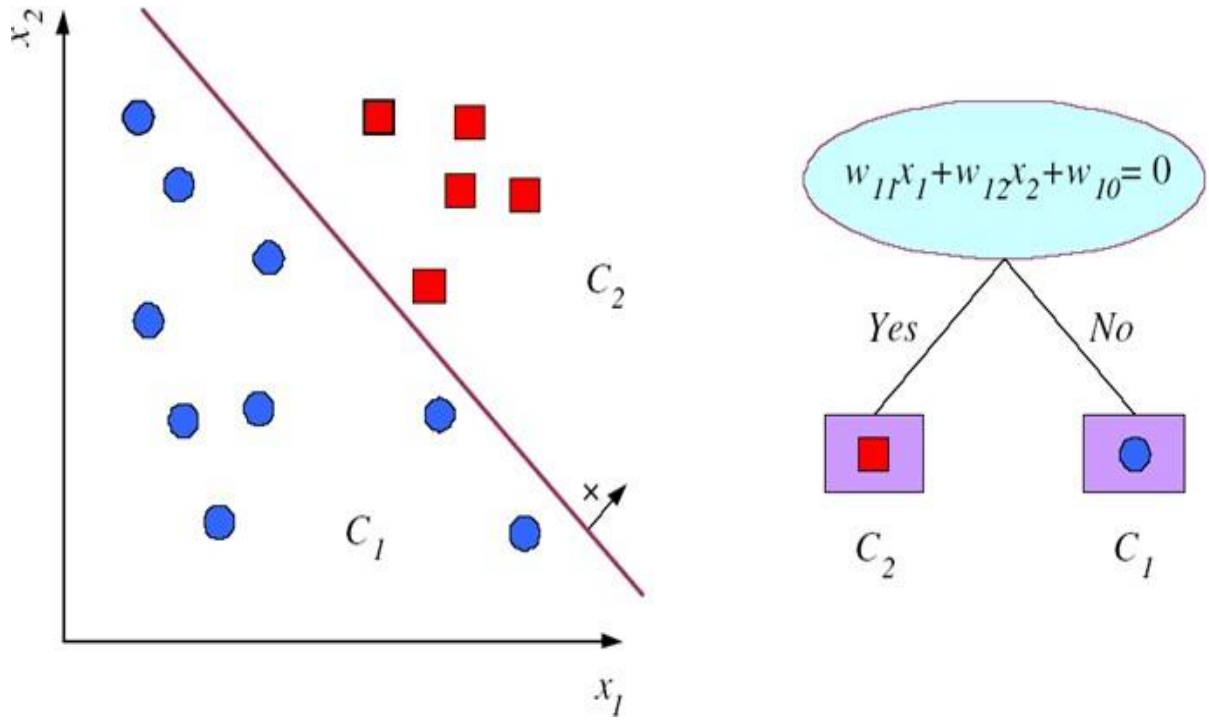
$$f_m(\boldsymbol{x}) : x_j > w_{m0}$$

where w_{m0} is a suitably chosen threshold value. The decision node divides the input space into two: $L_m = \{\boldsymbol{x}|x_j > w_{m0}\}$ and $R_m = \{\boldsymbol{x}|x_j \leq w_{m0}\}$; this is called a *binary split*. Successive decision nodes on a path from the root to a leaf further divide these into two using other attributes and generating splits orthogonal to each other. The leaf nodes define hyperrectangles in the input space (see figure 9.1).

Tree induction is the construction of the tree given a training sample. For a given training set, there exists many trees that code it with no error, and, for simplicity, we are interested in finding the smallest among them, where tree size is measured as the number of nodes in the tree and the complexity of the decision nodes. Finding the smallest tree is NP-complete (Quinlan 1986), and we are forced to use local search procedures based on heuristics that give reasonable trees in reasonable time.

Tree learning algorithms are greedy and, at each step, starting at the root with the complete training data, we look for the best split. This splits the training data into two or n, depending on whether the chosen attribute is numeric or discrete. We then continue splitting recursively with the corresponding subset until we do not need to split anymore, at which point a leaf node is created and labeled.

Multivariate Trees



Learning rules from data

From the decision tree, the following rules are identified.

If age=youth and student=no then
buys_computer=no. If age=youth and student=yes
then buys_computer=yes.If age=middle_aged then
buys_computer=yes.
If age=senior and credit_rating=fair then buys_computer=no.
If age=senior and credit_rating=excellent then buys_computer=yes.

- Rule induction is similar to tree induction but
  - tree induction is breadth-first,
  - rule induction is depth-first; one rule at a time
- Rule set contains rules; rules are conjunctions of terms
- Rule covers an example if all terms of the rule evaluate to true for the example
- Sequential covering: Generate rules one at a time until all positive examples are covered.

```
Ripper(Pos,Neg,k)
  RuleSet ← LearnRuleSet(Pos,Neg)
  For k times
    RuleSet ← OptimizeRuleSet(RuleSet,Pos,Neg)
LearnRuleSet(Pos,Neg)
  RuleSet ← ∅
  DL ← DescLen(RuleSet,Pos,Neg)
  Repeat
    Rule ← LearnRule(Pos,Neg)
    Add Rule to RuleSet
    DL' ← DescLen(RuleSet,Pos,Neg)
    If DL'>DL+64
      PruneRuleSet(RuleSet,Pos,Neg)
      Return RuleSet
    If DL'<DL DL ← DL'
      Delete instances covered from Pos and Neg
  Until Pos = ∅
  Return RuleSet
```

```
PruneRuleSet(RuleSet,Pos,Neg)
  For each Rule ∈ RuleSet in reverse order
    DL ← DescLen(RuleSet,Pos,Neg)
    DL' ← DescLen(RuleSet-Rule,Pos,Neg)
    IF DL'<DL Delete Rule from RuleSet
  Return RuleSet
OptimizeRuleSet(RuleSet,Pos,Neg)
  For each Rule ∈ RuleSet
      DL0 ← DescLen(RuleSet,Pos,Neg)
      DL1 ← DescLen(RuleSet-Rule+
       ReplaceRule(RuleSet,Pos,Neg),Pos,Neg)
      DL2 ← DescLen(RuleSet-Rule+
       ReviseRule(RuleSet,Rule,Pos,Neg),Pos,Neg)
      If DL1=min(DL0,DL1,DL2)
       Delete Rule from RuleSet and
          add ReplaceRule(RuleSet,Pos,Neg)
      Else If DL2=min(DL0,DL1,DL2)
       Delete Rule from RuleSet and
          add ReviseRule(RuleSet,Rule,Pos,Neg)
  Return RuleSet
```

Linear Discrimination

Likelihood- vs. Discriminant-based Classification:

- Likelihood-based: Assume a model for $p(\mathbf{x}|C_i)$, use Bayes' rule to calculate

  $P(C_i|\mathbf{x})g_i(\mathbf{x}) = \log P(C_i|\mathbf{x})$

- Discriminant-based: Assume a model for $g_i(\mathbf{x}|\Phi_i)$; no density estimation
- Estimating the boundaries is enough; no need to accurately estimate the densities insidethe boundaries

Linear Discriminant:

- Linear discriminant:

$$g_i\left(\mathbf{x}|\mathbf{w}_i, w_{i0}\right) = \mathbf{w}_i^T\mathbf{x} + w_{i0} = \sum_{j=1}^{d} w_{ij}x_j + w_{i0}$$

- Advantages:
  - Simple: O(d) space/computation
  - Knowledge extraction: Weighted sum of attributes; positive/negative weights, magnitudes (credit scoring)
  - Optimal when $p(\mathbf{x}|C_i)$ are Gaussian with shared cov matrix; useful when classes are (almost) linearly separable

Generalized Linear Model:

- Quadratic discriminant:

$$g_i\left(\mathbf{x}|\mathbf{W}_i, \mathbf{w}_i, w_{i0}\right) = \mathbf{x}^T\mathbf{W}_i\mathbf{x} + \mathbf{w}_i^T\mathbf{x} + w_{i0}$$

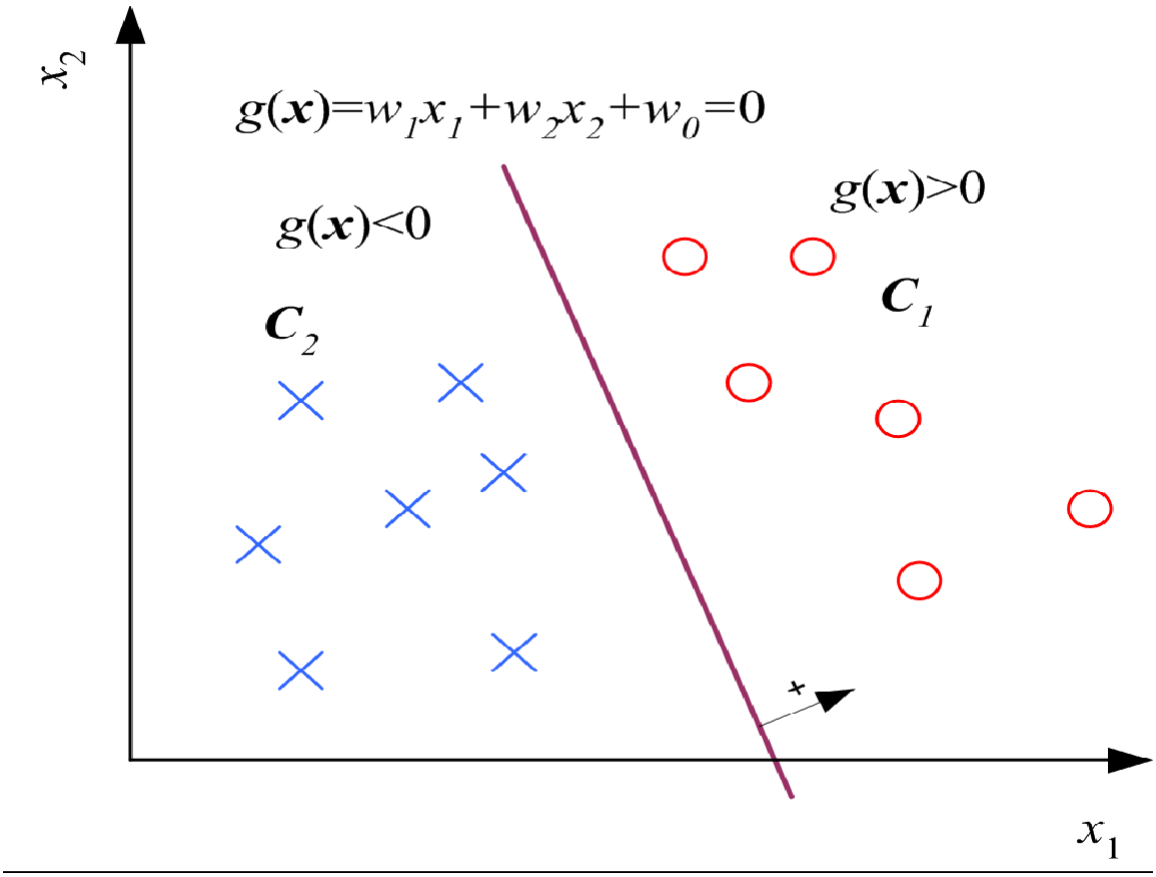
- Higher-order (product) terms:

$$z_1 = x_1,\ z_2 = x_2,\ z_3 = x_1^2,\ z_4 = x_2^2,\ z_5 = x_1x_2$$

- Map from x to z using nonlinear basis functions and use a linear discriminant in z-space:

$$g_i(\mathbf{x}) = \sum_{j=1}^{k} w_{ij}\phi_j(\mathbf{x})$$

Two Classes:

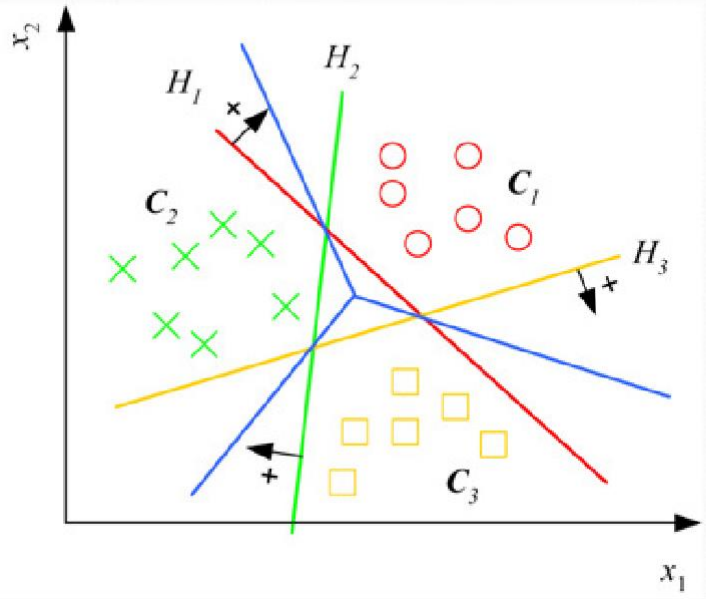


$$
\begin{aligned}
g(\mathbf{x}) &= g_1(\mathbf{x}) - g_2(\mathbf{x}) \\
&= (\mathbf{w}_1^T \mathbf{x} + w_{10}) - (\mathbf{w}_2^T \mathbf{x} + w_{20}) \\
&= (\mathbf{w}_1 - \mathbf{w}_2)^T \mathbf{x} + (w_{10} - w_{20}) \\
&= \mathbf{w}^T \mathbf{x} + w_0
\end{aligned}
$$

$$
\text{choose} \quad \begin{cases} C_1 & \text{if } g(\mathbf{x}) > 0 \\ C_2 & \text{otherwise} \end{cases}
$$

Multiple Classes:

$$g_i(\mathbf{x} \mid \mathbf{w}_i, w_{i0}) = \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

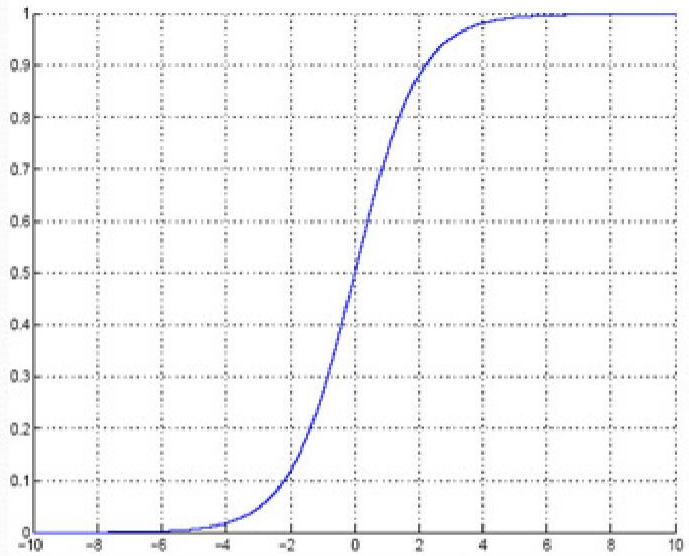


Choose C_i if

$$g_i(\mathbf{x}) = \max_{j=1}^{K} g_j(\mathbf{x})$$

Classes are linearly separable

Sigmoid (Logistic) Function:



1. Calculate $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$ and choose C_1 if $g(\mathbf{x}) > 0$, or
2. Calculate $y = \text{sigmoid}(\mathbf{w}^T \mathbf{x} + w_0)$ and choose C_1 if $y > 0.5$

Gradient-Descent:

- $E(\boldsymbol{w}|X)$ is error with parameters $\boldsymbol{w}$ on sample X

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} E(\boldsymbol{w} \mid X)$$

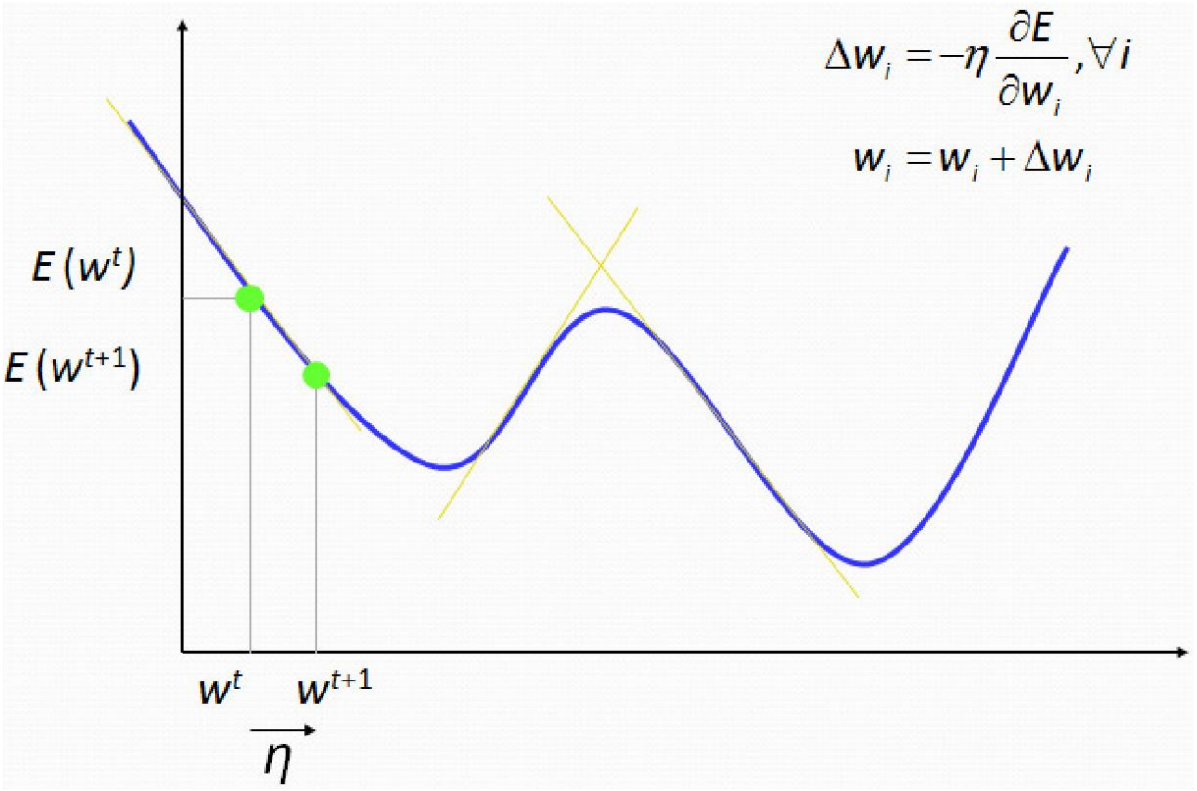
- Gradient

$$\nabla_w E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_d} \right]^T$$

- Gradient-descent:

  Starts from random $\boldsymbol{w}$ and updates $\boldsymbol{w}$ iteratively in the negative direction of gradient

---



Logistic Discrimination:

- Two classes: Assume log likelihood ratio is linear

$$\log\frac{p(\mathbf{x}\mid C_1)}{p(\mathbf{x}\mid C_2)} = \mathbf{w}^T\mathbf{x} + w_0^o$$

$$\text{logit}(P(C_1\mid\mathbf{x})) = \log\frac{P(C_1\mid\mathbf{x})}{1-P(C_1\mid\mathbf{x})} = \log\frac{p(\mathbf{x}\mid C_1)}{p(\mathbf{x}\mid C_2)} + \log\frac{P(C_1)}{P(C_2)}$$

$$= \mathbf{w}^T\mathbf{x} + w_0$$

$$\text{where } w_0 = w_0^o + \log\frac{P(C_1)}{P(C_2)}$$

$$y = \hat{P}(C_1\mid\mathbf{x}) = \frac{1}{1+\exp\left[-\left(\mathbf{w}^T\mathbf{x}+w_0\right)\right]}$$

TEXT / REFERENCE BOOKS

1. Chris Albon : Machine Learning with Python Cookbook , O"Reilly Media, Inc.2018
2. Stephen Marsland, "Machine Learning – An Algorithmic Perspective", Second Edition, Chapman and Hall/CRC Machine Learning and Pattern Recognition Series, 2014
3. Tom M. Mitchell, Machine Learning, India Edition 2013, McGraw Hill Education
4. Machine Learning: The art and Science of algorithms that make sense of data, Peter Flach, Cambridge University Press, 2012
5. EthemAlpaydın, Introduction to machine learning, second edition, MIT press.
6. T. Hastie, R. Tibshirani and J. Friedman, "Elements of Statistical Learning", Springer Series , 2nd edition
7. Sebastian Raschka, "Python Machine Learning",Second Edition.Packt Publication

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT-IV K-MEANS CLUSTERING

UNIT IV

UNIT IV K-MEANS CLUSTERING

Working of K-Means Clustering Algorithm, Advantages and Disadvantages, Applications of K-Means Clustering Algorithm, Hierarchical Clustering, Steps to Perform Agglomerative Hierarchical Clustering, Role of Dendrograms Agglomerative Hierarchical Clustering, Density-Based Clustering.

K-Means clustering:-

1) Decide the number & frame the clusters.
2) Find Mean of clusters.
3) Find distance b/w Mean & distance for all points.

Eg:- $x_1 = \{1, 0\}$, $x_2 = \{0, 1\}$, $x_3 = \{2, 1\}$, $x_4 = \{2, 3\}$.

Assume, we take two clusters, @

$C_1 = \{x_1, x_3\}$ & $C_2 = \{x_2, x_4\}$

(a) Apply 1 iteration of k-means partional clustering algorithm

(b) What is change in total square error.

(c) Apply 2nd iteration of k-means algorithm.

step 1:-

Mean,

$$M_k = \frac{1}{n_k} \sum_{i=1}^{k} x_{ik}$$

C_1:-

$x_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $x_3 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ $\Rightarrow M_1 = \left\{ \frac{1+2}{2}, \frac{0+1}{2} \right\}$

$= \{1.5, 0.5\}$.

$M_2 = \left\{ \frac{0+2}{2}, \frac{1+3}{2} \right\}$

$= \{1.5, 2\}$.

C_2 :- $x_3 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$ $x_4 = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$

Step 2 :-

Error Calculation.

$$e_k^2 = \sum_{i=1}^{k} (x_{ik} - M_{ik})^2$$

$M_1 = (1.5, 0.5)$ | $M_2 = (1.5, 2)$
$x_1 = (1, 0)$ | $x_3 = (0, 2)$
$x_2 = (2, 1)$ | $x_4 = (2, 3)$

$C_1 \Rightarrow e_1^2 = \left[(1-1.5)^2 + (0-0.5)^2 + (2-1.5)^2 + (1-0.5)^2 \right]$

$= 1.$

$C_2 \Rightarrow e_2^2 = \left[(0-1.5)^2 + (1-2)^2 + (3-1.5)^2 + (2-2)^2 \right]$

$= 6.5$

Step 3 :-

Total square error.

$$\Sigma_k^2 = \sum_{k=1}^{k} e_k^2$$

$\Rightarrow e_1^2 + e_2^2$

$1 + 6.5 = 7.5$

Step 4 :-

Distance calculation.

k-means follow euclidean's distance measure,

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

X_1 (1, 0):-

$d(X_1, M_1) = \sqrt{(1-1.5)^2 + (0-0.5)^2} = 0.707$

$d(X_1, M_2) = \sqrt{(1-1.5)^2 + (0-2)^2} = 2.062$

X_2 (0, 1):-

$d(X_2, M_1) = \sqrt{(0-1.5)^2 + (1-0.5)^2} = 1.581$

$d(X_2, M_2) = \sqrt{(0-1.5)^2 + (1-2)^2} = 1.803$

X_3 (2, 1):-

$d(X_3, M_1) = \sqrt{(2-1.5)^2 + (1-0.5)^2} = 0.707$

$d(X_3, M_2) = \sqrt{(2-1.5)^2 + (1-2)^2} = 1.118$

X_4 (3, 3):-

$d(X_4, M_1) = \sqrt{(3-1.5)^2 + (3-0.5)^2} = 2.915$

$d(X_4, M_2) = \sqrt{(3-1.5)^2 + (3-2)^2} = 1.8027$

	M_1	M_2
X_1	0.707	2.062
X_2	1.581	1.803
X_3	0.707	1.118
X_4	2.915	1.8027
	C_1	C_2

1) check the smallest element row wise

2) Divide the columns C_1 & C_2.

$\Rightarrow C_1 = (X_1, X_3, X_2)$

$C_2 = (X_4)$.

(c)

Mean,

$$M_i = \frac{1}{n_i} \sum_{i=1}^{r} x_{ij}$$

C_1:-

$x_1 = (1, 0)$

$x_2 = (0, 1)$ $\Rightarrow$ $M_1 = \left\{ \frac{1+0+0}{3}, \frac{0+1+1}{3} \right\}$

$x_3 = (2, 1)$ $= (1, 0.66)$

C_2:-

$x_4 = (3, 3)$ $\Rightarrow$ $M_2 = \{3, 3\}$.

Error Calculation,

$C_1 \Rightarrow e_1^2 \Rightarrow [(1-1)^2 + (0-0.66)^2 + (0-1)^2 + (1-0.66)^2 + (2-1)^2 + (1-0.66)^2]$

$= 0.4356 + 1 + 0.1156 + 1 + 0.1156 = 1.6668.$

$C_2 \Rightarrow e_2^2 \Rightarrow [(3-3)^2 + (3-3)^2] = 0.$

Total square error

$\Rightarrow e_1^2 + e_2^2 = 1.6668$

Distance Calculation,

$X_1 (1, 0)$

$d(x_1, M_1) = \sqrt{(1-1)^2 + (0-0.66)^2} = \sqrt{0 + 0.4356} = 0.66$

$d(x_2, M_2) = \sqrt{(1-1)^2 + (0-3)^2} = \sqrt{4+9} = \sqrt{13}$

$= 3.605$

$X_2\ (0,1)$

$$d(x_2, M_1) = \sqrt{(0-1)^2 + (0.666)(1-0.66)^2} = \sqrt{1+0.11} = 1.057$$

$$d(x_2, M_2) = \sqrt{(0-2)^2 + (1-3)^2} = \sqrt{9+4} = \sqrt{13} = 2.61$$

$X_3\ (2,1)$

$$d(x_3, M_1) = \sqrt{(2-1)^2 + (1-0.666)^2} = 1.056$$

$$d(x_3, M_2) = \sqrt{(2-3)^2 + (1-2)^2} = \sqrt{5} = 2.236$$

$X_4\ (3,3)$

$$d(x_4, M_1) = \sqrt{(3-1)^2 + (3-0.66)^2} = 9.4756.$$

$$d(x_4, M_2) = \sqrt{(3-3)^2 + (3-2)^2} = 0.$$

	M_1	M_2
X_1	0.66 ✓	3.605
X_2	1.056 ✓	2.605
X_3	1.056 ✓	2.236
X_4	9.4756	0 ✓
	C_1	C_2

$$\Rightarrow \quad C_1 = (x_1, x_3, x_2)$$

$$C_2 = (x_4).$$

The iteration stops here, as we get the same set of clusters as the previous step.

Advantages and Disadvantages

Advantages

The following are some advantages of K-Means clustering algorithms −

- It is very easy to understand and implement.
- If we have large number of variables then, K-means would be faster than Hierarchical clustering.
- On re-computation of centroids, an instance can change the cluster.
- Tighter clusters are formed with K-means as compared to Hierarchical clustering.

Disadvantages

The following are some disadvantages of K-Means clustering algorithms −

- It is a bit difficult to predict the number of clusters i.e. the value of k.
- Output is strongly impacted by initial inputs like number of clusters (value of k)
- Order of data will have strong impact on the final output.
- It is very sensitive to rescaling. If we will rescale our data by means of normalization or standardization, then the output will completely change.
- It is not good in doing clustering job if the clusters have a complicated geometric shape.

Applications of K-Means Clustering Algorithm

The main goals of cluster analysis are −

- To get a meaningful intuition from the data we are working with.
- Cluster-then-predict where different models will be built for different subgroups.

To fulfill the above-mentioned goals, K-means clustering is performing well enough. It can be used in following applications −

- Market segmentation
- Document Clustering
- Image segmentation
- Image compression
- Customer segmentation
- Analyzing the trend on dynamic data

Hierarchical Clustering

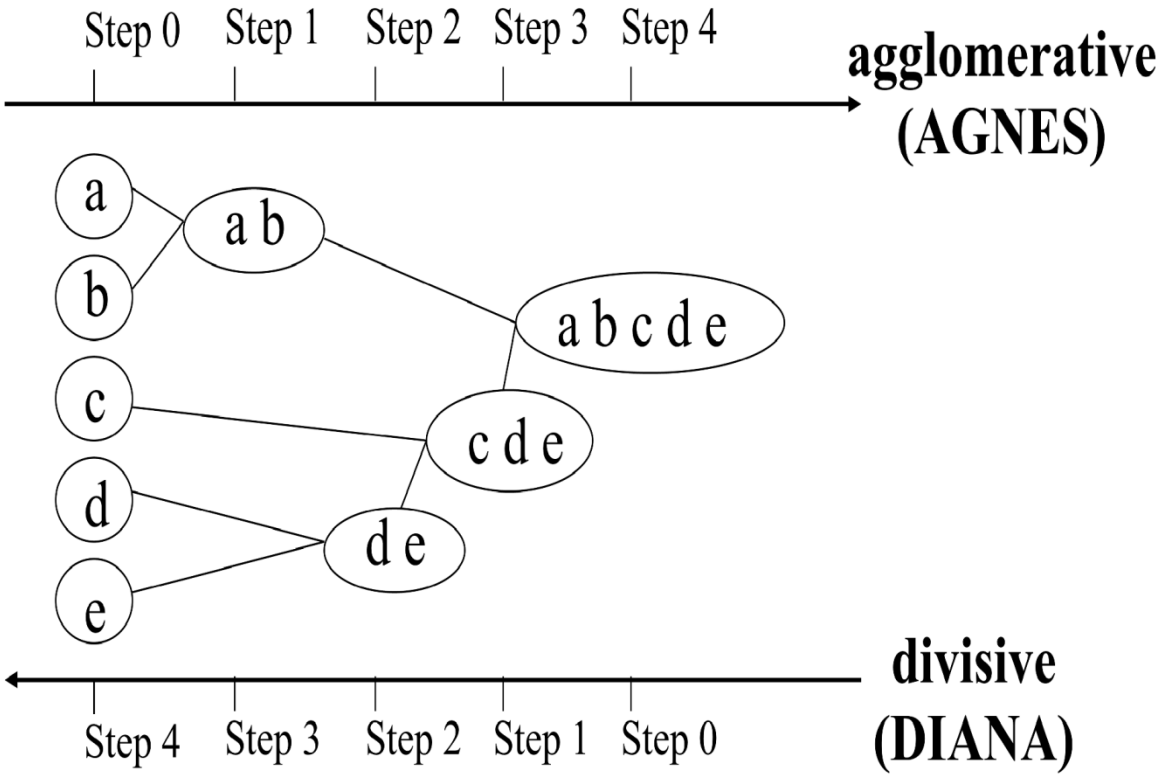
Cluster based on similarities/distances
Distance measure between instances xr and xs

Minkowski distance (Lp) (Euclidean for p = 2)

$$d_m(\mathbf{x}^r, \mathbf{x}^s) = \left[\sum_{j=1}^{d} \left(x_j^r - x_j^s\right)^p\right]^{1/p}$$

City-block distance

$$d_{cb}(\mathbf{x}^r, \mathbf{x}^s) = \sum_{j=1}^{d} \left|x_j^r - x_j^s\right|$$



- Two important paradigms:
  - Bottom-up agglomerative clustering(Agglomerative Nesting-AGNES)
  - Top-down divisive clustering(Divisive Analysis-DIANA)

Introduction to Hierarchical Clustering

Hierarchical clustering is another unsupervised learning algorithm that is used to group together the unlabeled data points having similar characteristics. Hierarchical clustering algorithms falls into following two categories.

Agglomerative hierarchical algorithms − In agglomerative hierarchical algorithms, each data point is treated as a single cluster and then successively merge or agglomerate (bottom-up approach) the pairs of clusters. The hierarchy of the clusters is represented as a dendrogram or tree structure.

Divisive hierarchical algorithms − On the other hand, in divisive hierarchical algorithms, all the data points are treated as one big cluster and the process of clustering involves dividing (Top-down approach) the one big cluster into various small clusters.

Steps to Perform Agglomerative Hierarchical Clustering

We are going to explain the most used and important Hierarchical clustering i.e. agglomerative. The steps to perform the same is as follows −

- **Step 1** − Treat each data point as single cluster. Hence, we will be having, say K clusters at start. The number of data points will also be K at start.
- **Step 2** − Now, in this step we need to form a big cluster by joining two closet datapoints. This will result in total of K-1 clusters.
- **Step 3** − Now, to form more clusters we need to join two closet clusters. This will result in total of K-2 clusters.

- **Step 4** − Now, to form one big cluster repeat the above three steps until K would become 0 i.e. no more data points left to join.
- **Step 5** − At last, after making one single big cluster, dendrograms will be used to divide into multiple clusters depending upon the problem.

Role of Dendrograms in Agglomerative Hierarchical Clustering

As we discussed in the last step, the role of dendrogram starts once the big cluster is formed. Dendrogram will be used to split the clusters into multiple cluster of related data points depending upon our problem.

What is Agglomerative Hierarchical Clustering

Agglomerative Hierarchical Clustering (**AHC**) is a clustering (or classification) method which has the following **advantages**:

It works from the dissimilarities between the objects to be grouped together. A type of dissimilarity can be suited to the subject studied and the nature of the data.

One of the results is the dendrogram which shows the progressive grouping of the data. It is then possible to gain an idea of a suitable number of classes into which the data can be grouped.

How does Agglomerative Hierarchical Clustering work

Agglomerative Hierarchical Clustering (AHC) is an iterative classification method whose principle is simple.

The process starts by calculating the dissimilarity between the N objects.

Then two objects which when clustered together minimize a given agglomeration criterion, are clustered together thus creating a class comprising these two objects.

Then the dissimilarity between this class and the N-2 other objects is calculated using the agglomeration criterion. The two objects or classes of objects whose clustering together minimizes the agglomeration criterion are then clustered together.

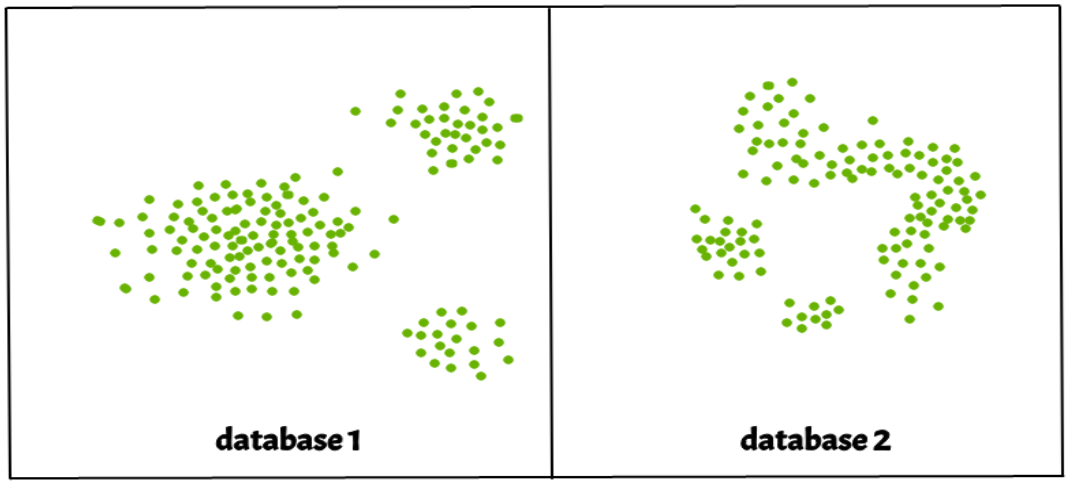
Agglomerative Hierarchical Clustering (AHC) is a clustering (or classification) method which has the following advantages: It works from the dissimilarities between the objects to be grouped together. A type of dissimilarity can be suited to the subject studied and the nature of the data. The **agglomerative clustering** is the most common type of hierarchical clustering used to group objects in clusters based on their similarity. It's also known as *AGNES* (*Agglomerative Nesting*). The algorithm starts by treating each object as a singleton cluster. Next, pairs of clusters are successively merged until all clusters have been merged into one big cluster containing all objects. The result is a tree-based representation of the objects, named *dendrogram*.

Clustering analysis or simply Clustering is basically an Unsupervised learning method that divides the data points into a number of specific batches or groups, such that the data points in the same groups have similar properties and data points in different groups have different properties in some sense. It comprises of many different methods based on different evolution. E.g. K-Means (distance between points), Affinity propagation (graph distance), Mean-shift (distance between points), DBSCAN (distance between nearest points), Gaussian mixtures (Mahalanobis distance to centers), Spectral clustering (graph distance) etc.

Fundamentally, all clustering methods use the same approach i.e. first we calculate similarities

and then we use it to cluster the data points into groups or batches. Here we will focus on **Density-based spatial clustering of applications with noise** (DBSCAN) clustering method.

Clusters are dense regions in the data space, separated by regions of the lower density of points. The **DBSCAN algorithm** is based on this intuitive notion of "clusters" and "noise". The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points.

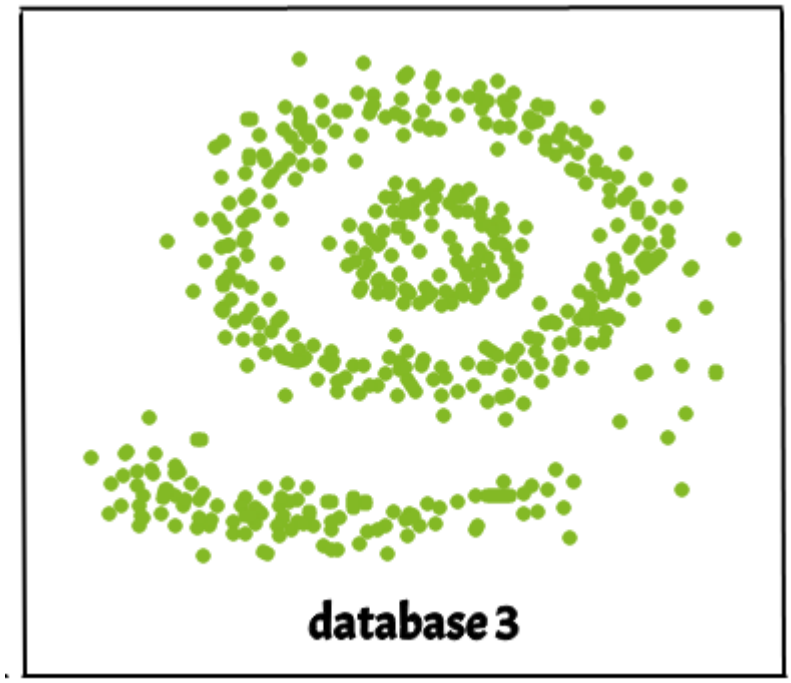


Partitioning methods (K-means, PAM clustering) and hierarchical clustering work for finding spherical-shaped clusters or convex clusters. In other words, they are suitable only for compact and well-separated clusters. Moreover, they are also severely affected by the presence of noise and outliers in the data.

Real life data may contain irregularities, like –

i) Clusters can be of arbitrary shape such as those shown in the figure below.

ii) Data may contain noise.



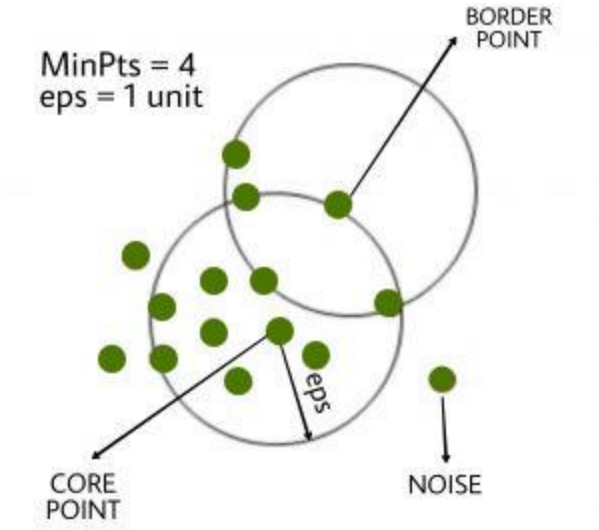
The figure below shows a data set containing nonconvex clusters and outliers/noises. Given such data, k-means algorithm has difficulties for identifying these clusters with arbitraryshapes.

DBSCAN algorithm requires two parameters –

1. **eps** : It defines the neighborhood around a data point i.e. if the distance between two points is lower or equal to 'eps' then they are considered as neighbors. If the eps value is chosen too small then large part of the data will be considered as outliers. If it is chosen very large then the clusters will merge and majority of the data points will be in the same clusters. One way to find the eps value is based on the ***k-distance graph***.
2. **MinPts**: Minimum number of neighbors (data points) within eps radius. Larger the dataset, the larger value of MinPts must be chosen. As a general rule, the minimum MinPts can be derived from the number of dimensions D in the dataset as, MinPts >= D+1. The minimum value of MinPts must be chosen at least 3.

In this algorithm, we have 3 types of data points.

Core Point: A point is a core point if it has more than MinPts points within eps.
Border Point: A point which has fewer than MinPts within eps but it is in the neighborhood of a core point.
Noise or outlier: A point which is not a core point or border point.



DBSCAN algorithm can be abstracted in the following steps –

1. Find all the neighbor points within eps and identify the core points or visited with more than MinPts neighbors.
2. For each core point if it is not already assigned to a cluster, create a new cluster.
3. Find recursively all its density connected points and assign them to the same cluster as the core point.

A point *a* and *b* are said to be density connected if there exist a point *c* which has a sufficient number of points in its neighbors and both the points *a* and *b* are within the *eps distance*. This is a chaining process. So, if *b* is neighbor of *c*, *c* is neighbor of *d*, *d* is neighbor of *e*, which in turn is neighbor of *a* implies that *b* is neighbor of *a*.

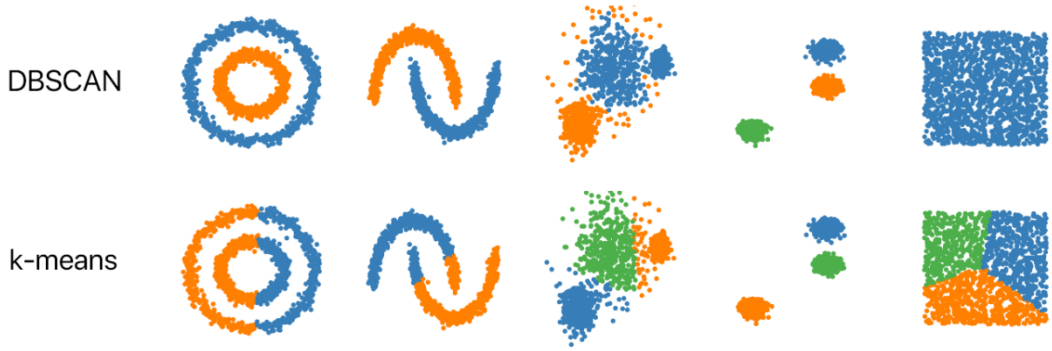
4. Iterate through the remaining unvisited points in the dataset. Those points that do not belong to any cluster are noise.

Why do we need a Density-Based clustering algorithm like DBSCAN when we already have K-means clustering?

K-Means clustering may cluster loosely related observations together. Every observation becomes a part of some cluster eventually, even if the observations are scattered far away in the vector space. Since clusters depend on the mean value of cluster elements, each data point plays a role in forming the clusters. A slight change in data points *might* affect the clustering outcome. This problem is greatly reduced in DBSCAN due to the way clusters are formed. This is usually not a big problem unless we come across some odd shape data.

Another challenge with *k*-means is that you need to specify the number of clusters ("*k*") in order to use it. Much of the time, we won't know what a reasonable *k* value is *a priori*.

What's nice about DBSCAN is that you don't have to specify the number of clusters to use it. All you need is a function to calculate the distance between values and some guidance for what amount of distance is considered "close". DBSCAN also produces more reasonable results than *k*-means across a variety of different distributions. Below figure illustrates the fact:



Density-Based Clustering Algorithms

Density-Based **Clustering** refers to unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.

The DBSCAN algorithm uses two parameters:

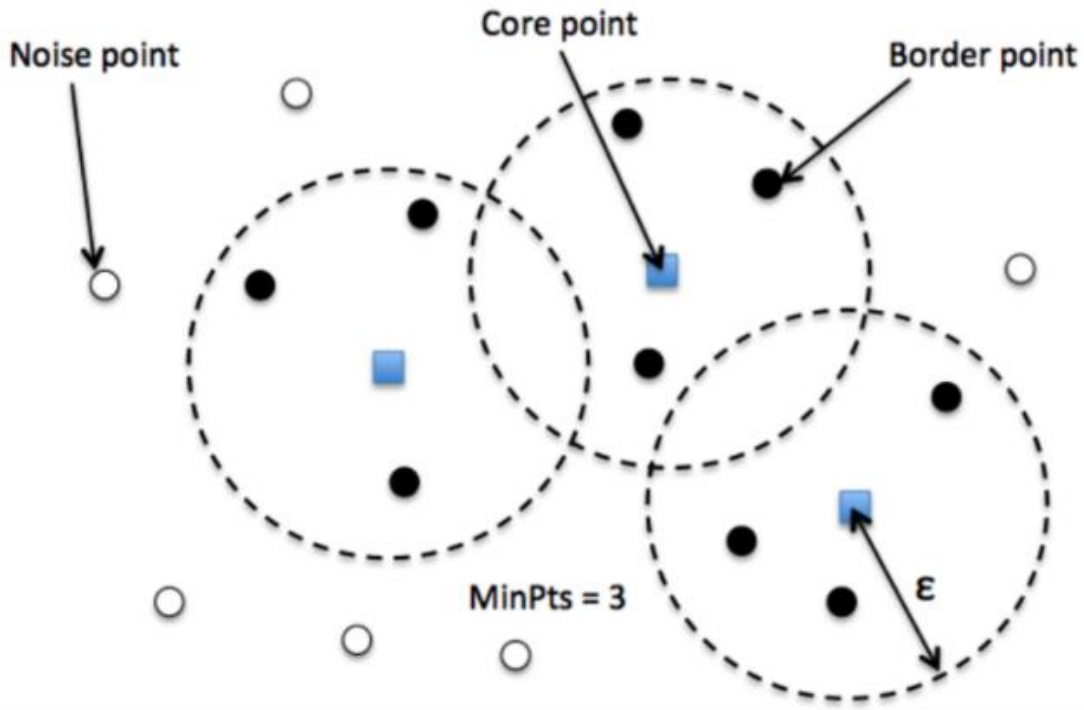
- **minPts:** The minimum number of points (a threshold) clustered together for a region to be considered dense.
- **eps (ε):** A distance measure that will be used to locate the points in the neighborhood of any point.

These parameters can be understood if we explore two concepts called Density Reachability and Density Connectivity.

Reachability in terms of density establishes a point to be reachable from another if it lies within a particular distance (eps) from it.

Connectivity, on the other hand, involves a transitivity based chaining-approach to determine whether points are located in a particular cluster. For example, p and q points could be connected if p->r->s->t->q, where a->b means b is in the neighborhood of a.

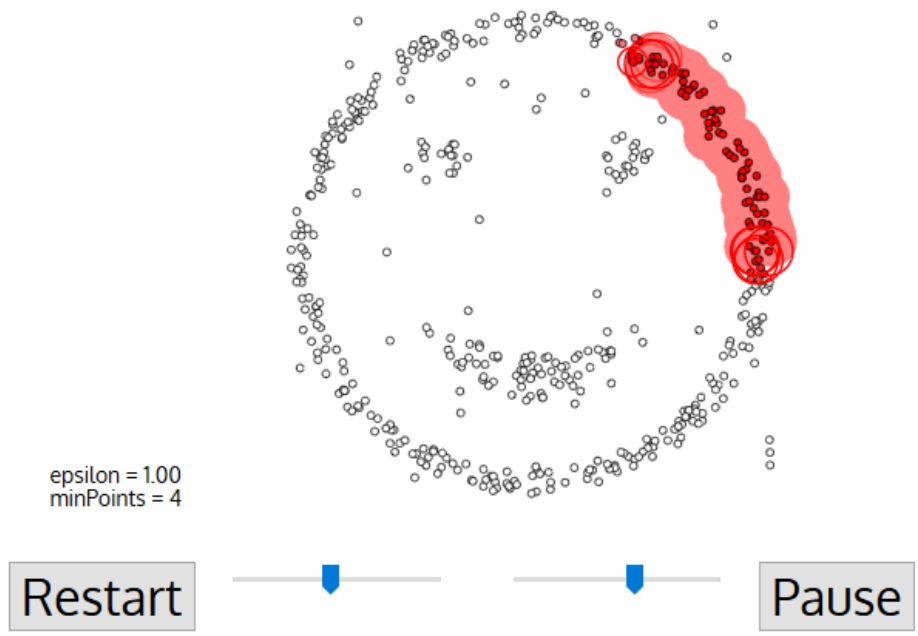
There are three types of points after the DBSCAN clustering is complete:



- **Core** — This is a point that has at least *m* points within distance *n* from itself.
- **Border** — This is a point that has at least one Core point at a distance *n*.
- **Noise** — This is a point that is neither a Core nor a Border. And it has less than *m* points within distance *n* from itself.

Algorithmic steps for DBSCAN clustering

- The algorithm proceeds by arbitrarily picking up a point in the dataset (until all points have been visited).
- If there are at least 'minPoint' points within a radius of 'ε' to the point then we consider all these points to be part of the same cluster.
- The clusters are then expanded by recursively repeating the neighborhood calculation for each neighboring point



DBSCAN in action

Parameter Estimation

Every data mining task has the problem of parameters. Every parameter influences the algorithm in specific ways. For DBSCAN, the parameters **ε** and **minPts** are needed.

- **minPts**: As a rule of thumb, a minimum *minPts* can be derived from the number of dimensions *D* in the data set, as $\boldsymbol{minPts \geq D + 1}$. The low value $\boldsymbol{minPts = 1}$ does not make sense, as then every point on its own will already be a cluster. With $\boldsymbol{minPts \leq 2}$, the result will be the same as of hierarchical clustering with the single link metric, with the dendrogram cut at height ε. Therefore, *minPts* must be chosen at least 3. However, larger values are usually better for data sets with noise and will yield more significant clusters. As a rule of thumb, $\boldsymbol{minPts = 2 \cdot dim}$ can be used, but it may be necessary to choose larger values for very large data, for noisy data or for data that contains many duplicates.
- **ε**: The value for ε can then be chosen by using a k-distance graph, plotting the distance to the $\boldsymbol{k = minPts - 1}$ nearest neighbor ordered from the largest to the smallest value. Good values of ε are where this plot shows an "elbow": if ε is chosen much too small, a large part of the data will not be clustered; whereas for a too high value of ε, clusters will merge and the majority of objects will be in the same cluster. In general, small values of ε are preferable, and as a rule of thumb, only a small fraction of points should be within this distance of each other.
- **Distance function**: The choice of distance function is tightly linked to the choice of ε, and has a major impact on the outcomes. In general, it will be necessary to first identify a reasonable measure of similarity for the data set, before the parameter ε can be chosen. There is no estimation for this parameter, but the distance functions need to be chosen appropriately for the data set.

TEXT / REFERENCE BOOKS

1. Chris Albon : Machine Learning with Python Cookbook , O"Reilly Media, Inc.2018
2. Stephen Marsland, "Machine Learning – An Algorithmic Perspective", Second Edition, Chapman and Hall/CRC Machine Learning and Pattern Recognition Series, 2014
3. Tom M. Mitchell, Machine Learning, India Edition 2013, McGraw Hill Education
4. Machine Learning: The art and Science of algorithms that make sense of data, Peter Flach, Cambridge University Press, 2012
5. EthemAlpaydın, Introduction to machine learning, second edition, MIT press.
6. T. Hastie, R. Tibshirani and J. Friedman, "Elements of Statistical Learning", Springer Series , 2nd edition
7. Sebastian Raschka, "Python Machine Learning",Second Edition.Packt Publication

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT V- DIMENSIONALITY REDUCTIONS & COLLABORATIVE FILTERING

UNIT V

UNIT V DIMENSIONALITY REDUCTIONS & COLLABORATIVE FILTERING

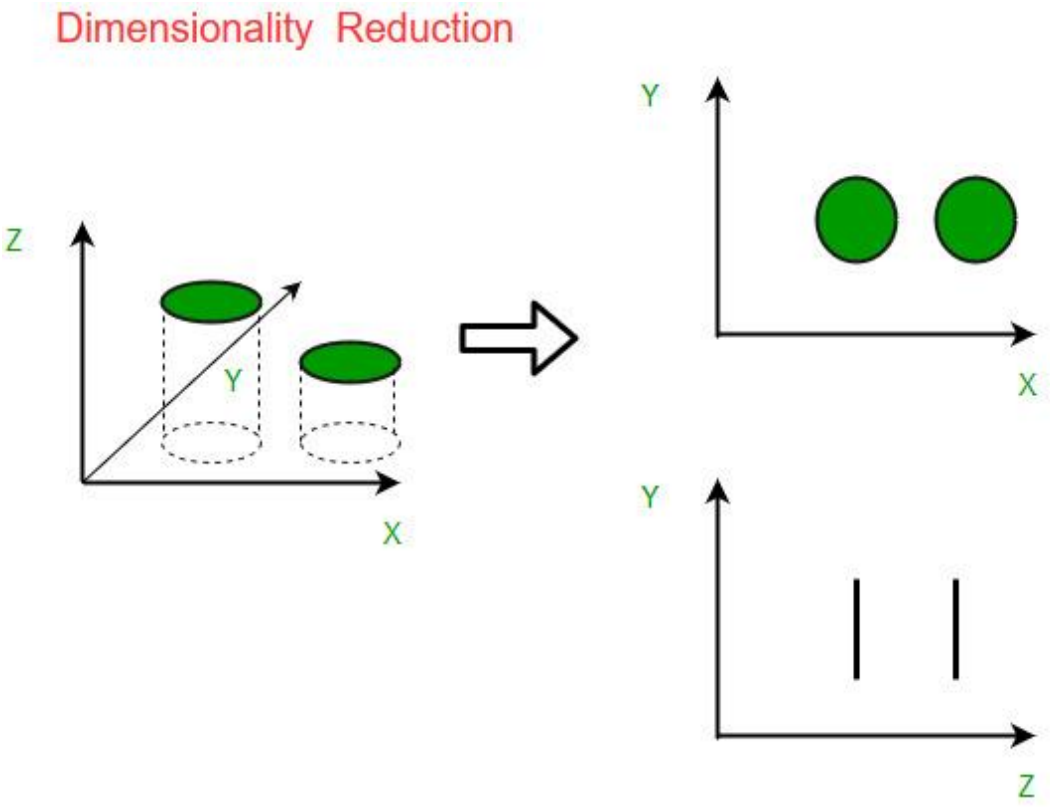
Dimensionality Reduction, Feature Extraction & Selection, Linear Discriminant Analysis, Principal Component Analysis, Factor Analysis, Independent Component Analysis, Locally Linear Embedding, Least Squares Optimization, Collaborative Filtering & Its Challenges.

What is Dimensionality Reduction?

In machine learning classification problems, there are often too many factors on the basis of which the final classification is done. These factors are basically variables called features. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

Why is Dimensionality Reduction important in Machine Learning and Predictive Modeling?

An intuitive example of dimensionality reduction can be discussed through a simple e-mail classification problem, where we need to classify whether the e-mail is spam or not. This can involve a large number of features, such as whether or not the e-mail has a generic title, the content of the e-mail, whether the e-mail uses a template, etc. However, some of these features may overlap. In another condition, a classification problem that relies on both humidity and rainfall can be collapsed into just one underlying feature, since both of the aforementioned are correlated to a high degree. Hence, we can reduce the number of features in such problems. A 3-D classification problem can be hard to visualize, whereas a 2-D one can be mapped to a simple 2 dimensional space, and a 1-D problem to a simple line. The below figure illustrates this concept, where a 3-D feature space is split into two 1-D feature spaces, and later, if found to be correlated, the number of features can be reduced even further.



Components of Dimensionality Reduction
There are two components of dimensionality reduction:

- **Feature selection:** In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem. It usually involves three ways:

1. Filter
2. Wrapper
3. Embedded

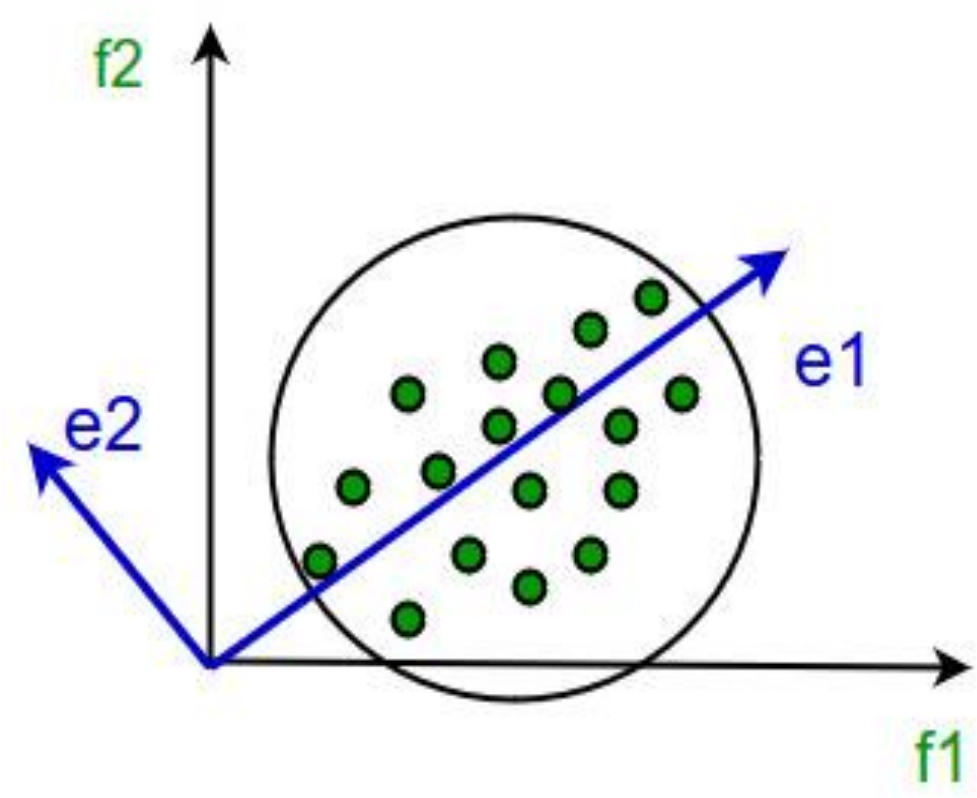
- **Feature extraction:** This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

Methods of Dimensionality Reduction
The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

Dimensionality reduction may be both linear or non-linear, depending upon the method used. The prime linear method, called Principal Component Analysis, or PCA, is discussed below.

Principal Component Analysis
This method was introduced by Karl Pearson. It works on a condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.



It involves the following steps:

- Construct the covariance matrix of the data.
- Compute the eigenvectors of this matrix.
- Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction of variance of the original data.

Hence, we are left with a lesser number of eigenvectors, and there might have been some data loss in the process. But, the most important variances should be retained by the remaining eigenvectors.

Advantages of Dimensionality Reduction

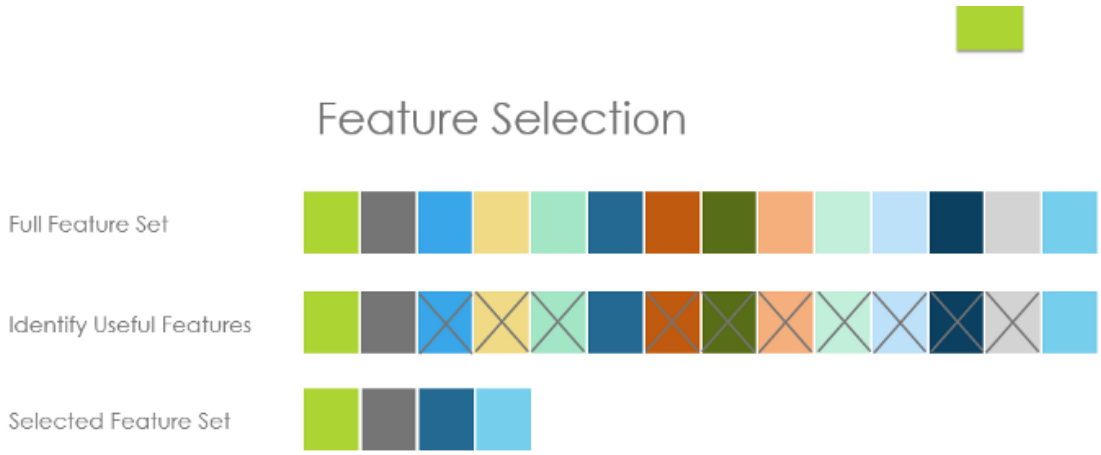
- It helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any.

Disadvantages of Dimensionality Reduction

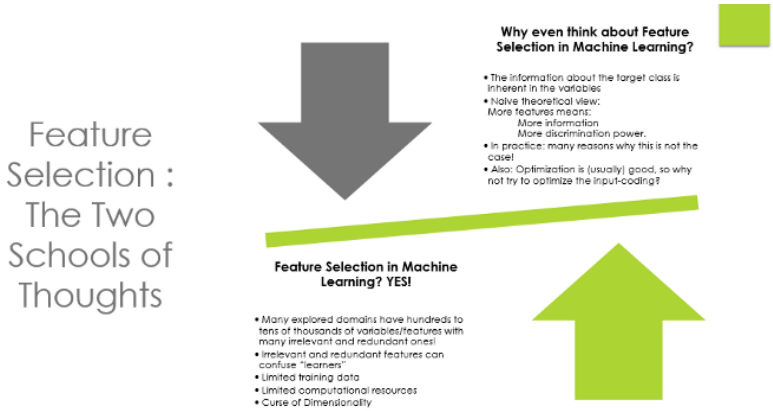
- It may lead to some amount of data loss.
- PCA tends to find linear correlations between variables, which is sometimes undesirable.
- PCA fails in cases where mean and covariance are not enough to define datasets.
- We may not know how many principal components to keep- in practice, some thumb rules are applied.

What is Feature selection (or Variable Selection)?

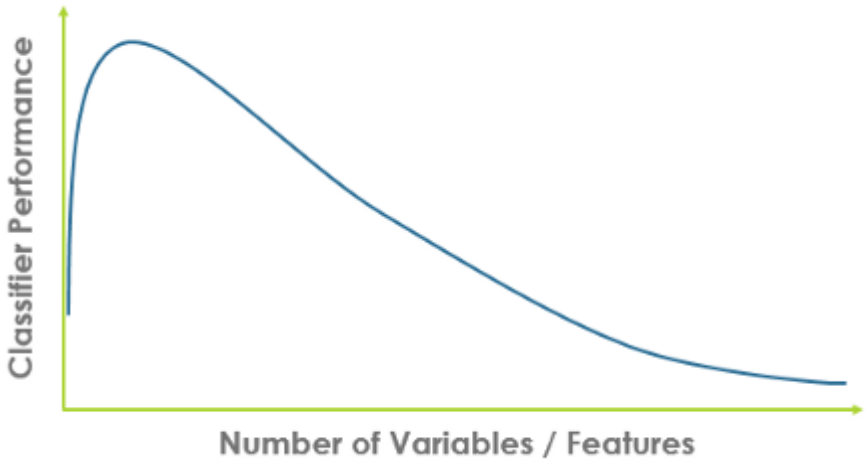
Problem of selecting some subset of a learning algorithm's input variables upon which it should focus attention, while ignoring the rest. In other words, **Dimensionality Reduction.** As Humans, we constantly do that!



What is Feature selection (or Variable Selection)?
Mathematically speaking,
Given a set of features $F = \{ f1 , ..., fi , ..., fn \}$ the Feature Selection problem is to find a subset that "maximizes the learner's ability to classify patterns"
Formally F' should maximize some scoring function
This general definition subsumes **feature selection** (i.e. a **feature selection algorithm** also performs a mapping but can only map to subsets of the input variables)
Feature Selection : The Two Schools of Thoughts
There are two schools of thoughts on **Feature Selection.**



Feature Selection : The Two Schools of Thoughts
While the above two school of thoughts co-exist, we'd focus here on the Motivation for **Feature Selection.**
Especially when dealing with a large number of variables there is a need for **Dimensionality Reduction**
Feature Selection can significantly improve a learning algorithm's performance
The Curse of Dimensionality



The Curse of Dimensionality

The required number of samples (to achieve the same accuracy) grows exponentially with the number of variables.

In practice: the number of training examples is fixed.

the classifier's performance usually will degrade for a large number of features

In many cases the information that is lost by discarding variables is made up for by a more accurate mapping/sampling in the lower-dimensional space.

Feature Selection — Optimality?

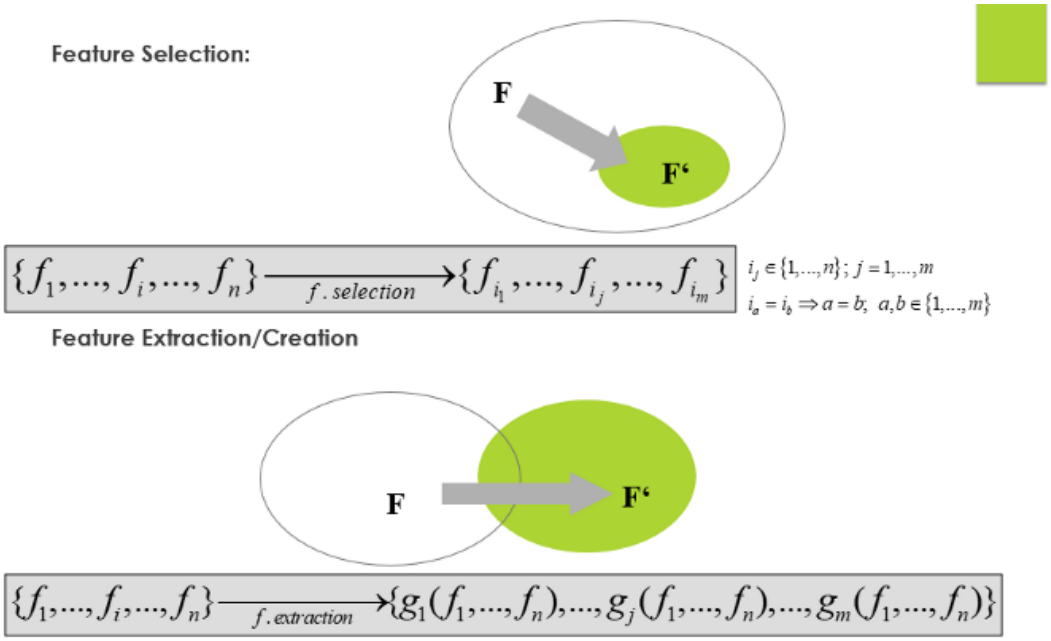
In theory, the goal is to find an optimal feature-subset (one that maximizes the scoring function).

In real world applications this is usually not possible.

For most problems it is computationally intractable to search the whole space of possible feature subsets

One usually has to settle for approximations of the optimal subset

Most of the research in this area is devoted to finding efficient search-heuristics



Feature Selection and Feature Extraction

Optimal Feature Subset:

Often, the definition of optimal feature subset in terms of classifier's performance

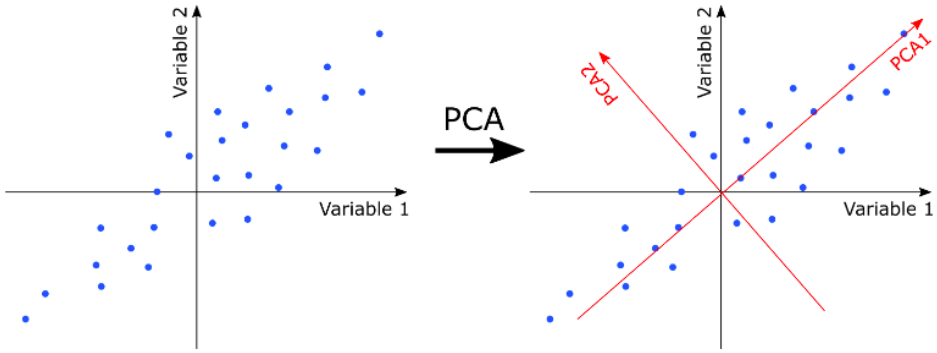
The best one can hope for theoretically is the Bayes error rate

Relevance of Features

There are several definitions of relevance in literature.
Relevance of 1 variable, Relevance of a variable given other variables, Relevance given a certain learning algorithm ,..
Most definitions are problematic, because there are problems where all features would be declared to be irrelevant
This can be defined through two degrees of relevance: **weak and strong relevance**.
A feature is relevant *iff* it is weakly or strongly relevant and irrelevant (redundant) otherwise.
Strong Relevance of a variable/feature:
Let $Si = \{f1, ..., fi-1, fi+1, ...fn\}$ be the set of all features except *fi*. Denote by *si* a value-assignment to all features in *Si*.
A feature *fi* is strongly relevant, iff there exists some *xi*, *y* and *si* for which $p(fi = xi, Si = si) > 0$ such that
$p(Y = y \mid fi = xi; Si = si) \neq p(Y = y \mid Si = si)$
This means that removal of *fi* alone will always result in a performance deterioration of an optimal Bayes classifier.
Weak Relevance of a variable/feature:
A feature fi is weakly relevant, iff it is not strongly relevant, and there exists a subset of features Si‘ of Si for which there exists some xi, y and si’ with p(fi = xi, Si’ = si’) > 0 such that
$p(Y = y \mid fi = xi; Si' = si') \neq p(Y = y \mid Si' = si')$
This means that there exists a subset of features *Si’*, such that the performance of an optimal Bayes classifier on *Si’* is worse than *Si’* **U** { *fi* }

Feature extraction is about **extracting/deriving** information from the original features set to create a new features subspace. The primary idea behind feature extraction is to compress the data with the goal of maintaining most of the relevant information. As with feature selection techniques, these techniques are also used for reducing the number of features from the original features set to reduce model complexity, model overfitting, enhance model computation efficiency and reduce generalization error. The following are different types of feature extraction techniques:

Principal component analysis (**PCA**) for unsupervised data compression. Here is a detailed post on feature extraction using PCA with Python example. You will get a good understanding of how PCA can help with finding the directions of maximum variance in high-dimensional data and projects the data onto a new subspace with equal or fewer dimensions than the original one**.** This is explained with example of identifying **Taj Mahal (7th wonder of world)** from top view or side view based on **dimensions** in which there is **maximum variance**. The diagram below shows the dimensions of maximum variance (PCA1 and PCA2) as a result of PCA.



Linear discriminant analysis (**LDA**) as a supervised dimensionality reduction technique for maximizing class separability

Nonlinear dimensionality reduction via **kernel principal component analysis** (**KPCA**)

When to use Feature Selection & Feature Extraction

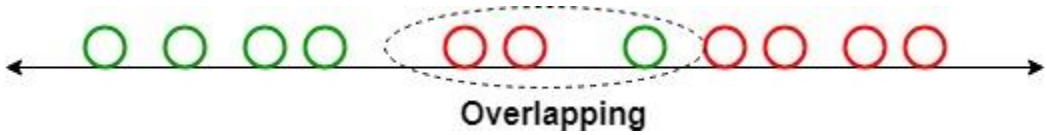
The **key difference** between feature selection and feature extraction techniques used for dimensionality reduction is that while the **original features are maintained** in the case of feature selection algorithms, the feature extraction algorithms **transform the data onto a new feature space**.

Feature selection techniques can be used if the requirement is to **maintain the original features,** unlike the feature extraction techniques which derive useful information from data to construct a new feature subspace. Feature selection techniques are used when model explainability is a key requirement. Feature extraction techniques can be used to improve the predictive performance of the models, especially, in the case of algorithms that don't support regularization.

Unlike feature selection, feature extraction usually needs to transform the original data to features with strong pattern recognition ability, where the original data can be regarded as features with weak recognition ability.

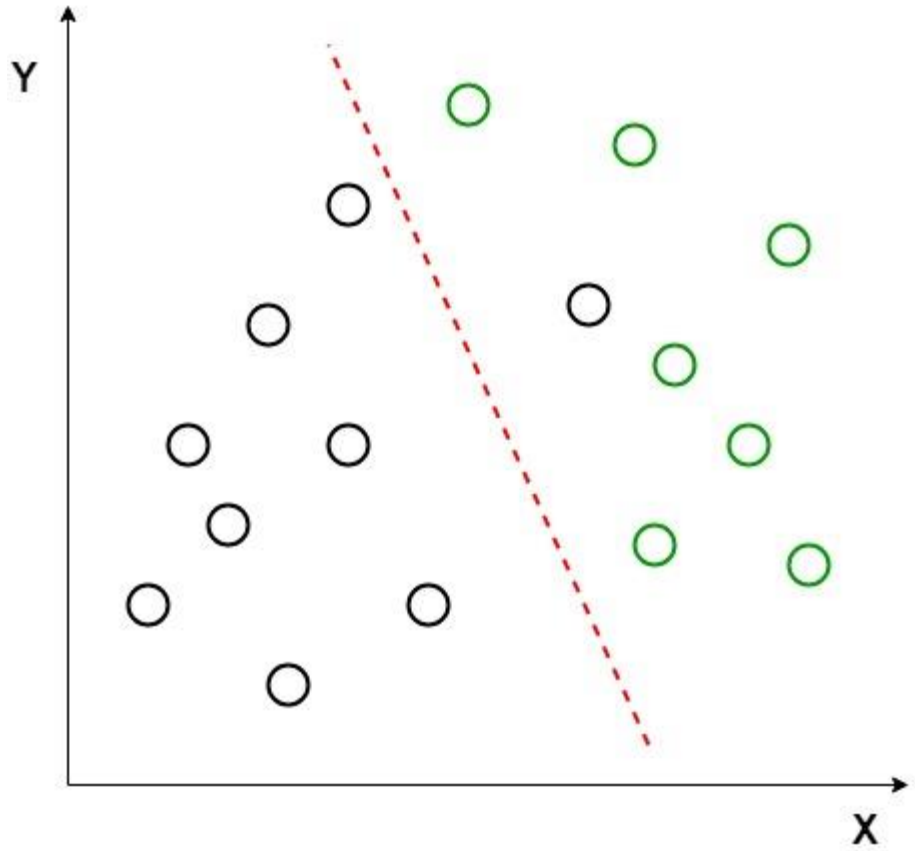
Linear Discriminant Analysis or Normal Discriminant Analysis or Discriminant Function Analysis is a dimensionality reduction technique which is commonly used for the supervised classification problems. It is used for modeling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space.

For example, we have two classes and we need to separate them efficiently. Classes can have multiple features. Using only a single feature to classify them may result in some overlapping as shown in the below figure. So, we will keep on increasing the number of features for proper classification.



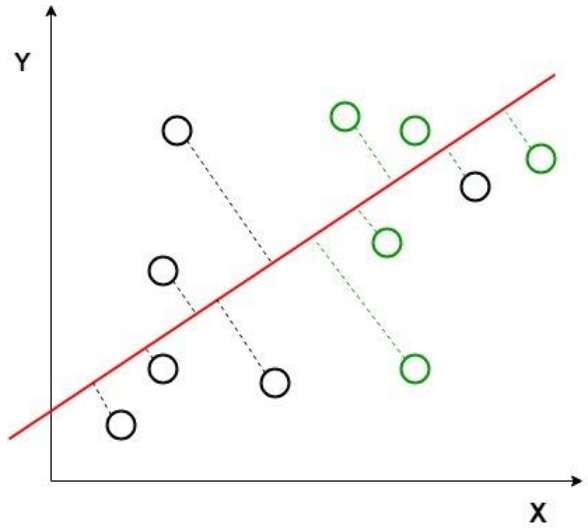
Example:

Suppose we have two sets of data points belonging to two different classes that we want to classify. As shown in the given 2D graph, when the data points are plotted on the 2D plane, there's no straight line that can separate the two classes of the data points completely. Hence, in this case, LDA (Linear Discriminant Analysis) is used which reduces the 2D graph into a 1D graph in order to maximize the separability between the two classes.

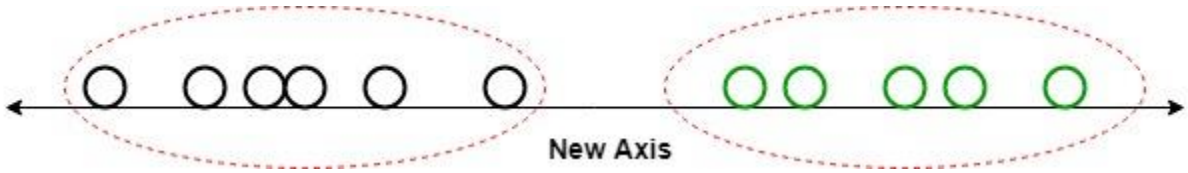


Here, Linear Discriminant Analysis uses both the axes (X and Y) to create a new axis and projects data onto a new axis in a way to maximize the separation of the two categories and hence, reducing the 2D graph into a 1D graph.
Two criteria are used by LDA to create a new axis:

1. Maximize the distance between means of the two classes.
2. Minimize the variation within each class.



In the above graph, it can be seen that a new axis (in red) is generated and plotted in the 2D graph such that it maximizes the distance between the means of the two classes and minimizes the variation within each class. In simple terms, this newly generated axis increases the separation between the data points of the two classes. After generating this new axis using the above-mentioned criteria, all the data points of the classes are plotted on this new axis and are shown in the figure given below.



But Linear Discriminant Analysis fails when the mean of the distributions are shared, as it becomes impossible for LDA to find a new axis that makes both the classes linearly separable. In such cases, we use non-linear discriminant analysis.

Mathematics

Let's suppose we have two classes and a d- dimensional samples such as x1, x2 … xn, where: n1 samples coming from the class (c1) and n2 coming from the class (c2).

If xi is the data point, then its projection on the line represented by unit vector v can be written as vTxi

Let's consider u1 and u2 be the means of samples class c1 and c2 respectively before projection and u1hat denotes the mean of the samples of class after projection and it can be calculated by:

$\widetilde{\mu_1} = \frac{1}{n_1} \sum_{x_i \in c_1}^{n_1} v^T x_i = v^T \mu_1$

Similarly,

$\widetilde{\mu_2} = v^T \mu_2$

Now, In LDA we need to normalize |\widetilde{\mu_1} -\widetilde{\mu_2} |. Let y_i = v^{T}x_i be the projected samples, then scatter for the samples of c1 is:

$\widetilde{s_1^2} = \sum_{y_i \in c_1} (y_i - \mu_1)^2$

Similarly:

$\widetilde{s_2^2} = \sum_{y_i \in c_1} (y_i - \mu_2)^2$

Now, we need to project our data on the line having direction v which maximizes

$J(v) = \frac{\widetilde{\mu_1} - \widetilde{\mu_2}}{\widetilde{s_1^2} + \widetilde{s_2^2}}$

For maximizing the above equation we need to find a projection vector that maximizes the difference of means of reduces the scatters of both classes. Now, scatter matrix of s1 and s2 of classes c1 and c2 are:

$$s_1 = \sum_{x_i \in c_1} (x_i - \mu_1)(x_i - \mu_1)^T$$

and s2

$$s_2 = \sum_{x_i \in c_2} (x_i - \mu_2)(x_i - \mu_2)^T$$

After simplifying the above equation, we get:

Now, we define, scatter within the classes(s_w) and scatter b/w the classes(s_b):

$$s_w = s_1 + s_2$$

$$s_b = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$$

Now, we try to simplify the numerator part of J(v)

$$J(v) = \frac{|\widetilde{\mu_1} - \widetilde{\mu_2}|}{\widetilde{s_1^2} + \widetilde{s_2^2}} = \frac{v^T s_b v}{v^T s_w v}$$

Now, To maximize the above equation we need to calculate differentiation with respect to v

$$\frac{dJ(v)}{dv} = s_b v - \frac{v^t s_b v (s_w v)}{v^T s_w v}$$

$$= s_b v - \lambda s_w v = 0$$

$$s_b v = \lambda s_w v$$

$$s_w^{-1} s_b v = \lambda v$$

$$Mv = \lambda v$$

$$where,$$

$$\lambda = \frac{v^T s_b v}{v^T s_w v} and$$

$$M = s_w^{-1} s_b$$

Here, for the maximum value of J(v) we will use the value corresponding to the highest eigenvalue. This will provide us the best solution for LDA.

ExtensionstoLDA:

1. **Quadratic Discriminant Analysis (QDA):** Each class uses its own estimate of variance (or covariance when there are multiple input variables).
2. **Flexible Discriminant Analysis (FDA):** Where non-linear combinations of inputs is used such as splines.
3. **Regularized Discriminant Analysis (RDA):** Introduces regularization into the estimate of the variance (actually covariance), moderating the influence of different variables on LDA.

What is linear discriminant analysis?

It is one of the most used dimensionality reduction techniques. It is used in machine learning as well as applications that have anything to do with the classification of patterns. LDA serves a very specific purpose, which is to project features that exist in a high dimensional space onto space at a lower dimension.

This is done to do away with common dimensionality issues and bring down dimensional costs and resources. Ronald A Fisher holds the credit for the development of the original concept in **1936 –Fisher's Discriminant Analysis or Linear Discriminant**. Originally, linear discriminant was a two-class technique. The multi-class version came in later.

Linear discriminant analysis is a supervised classification method that is used to create machine learning models. These models based on dimensionality reduction are used in the application, such as marketing predictive analysis and image recognition, amongst others. We will discuss applications a little later.

So what are we exactly looking for with LDA? There are two areas that this dimensionality reduction technique helps in discovering – The parameters that can be used to explain the relationship between a group and an object – The classification preceptor model that can help in separating the groups. This is why LDA is widely used to model varieties in different groups. So you can use this technique to use two or more than two classes for the distribution of a variable.

Extensions to linear discriminant analysis

LDA is considered one of the simplest and most effective methods available for classification. As the method is so simple and easy to understand, we have a few variations as well as extensions available for it. Some of these include:

1. Regularized discriminant analysis or RDA

RDA is used for bringing regularization into variance or covariance estimation. This is done to moderate the impact that variables have on LDA.

2. Quadratic discriminant analysis or QDA

In QDA, different classes use their own variance estimate. In case the number of the input variable is more than usual, every class uses its covariance estimate.

3. Flexible discriminant analysis or FDA

FDA makes use of inputs with non-linear combinations. Splines are a good example.

Common LDA applications

LDA finds its use in several applications. It can be used in any problem that can be turned into a classification problem. Common examples include speed recognition, face recognition, chemistry, microarray data classification, image retrieval, biometrics, and bioinformatics to name a few. Let's discuss a few of these.

1. Face recognition

In computer vision, face recognition is considered one of the most popular applications. Face recognition is carried out by representing faces using large amounts of pixel values. LDA is used to trim down the number of features to prepare grounds for using the classification method. The new dimensions are combinations of pixel values that are used to create a template.

2. Customer identification

If you want to identify customers on the basis of the likelihood that they will buy a product, you can use LDA to collect customer features. You can identify and choose those features that describe the group of customers that are showing higher chances of buying a product.

3. Medical

LDA can be used to put diseases into different categories, such as severe, mild, or moderate. There are several patient parameters that will go into conducting this classification task. This classification allows doctors to define the pace of the treatment.

Principal Component Analysis

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are ***image processing, movie recommendation system, optimizing the power allocation in various communication channels.*** It is a feature extraction technique, so it contains the important variables and drops the least important variable.

The PCA algorithm is based on some mathematical concepts such as:

- Variance and Covariance
- Eigenvalues and Eigen factors

Some common terms used in PCA algorithm:

- **Dimensionality:** It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.
- **Correlation:** It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
- **Orthogonal:** It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.
- **Eigenvectors:** If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.
- **Covariance Matrix:** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

Principal Components in PCA

As described above, the transformed new features or the output of PCA are the Principal Components. The number of these PCs are either equal to or less than the original features present in the dataset. Some properties of these principal components are given below:

- The principal component must be the linear combination of the original features.
- These components are orthogonal, i.e., the correlation between a pair of variables is zero.
- The importance of each component decreases when going to 1 to n, it means the 1 PC has the most importance, and n PC will have the least importance.

Steps for PCA algorithm

1. **Getting the dataset**

Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.

2. **Representing data into a structure**

Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.

3. **Standardizing the data**

In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance.

If the importance of features is independent of the variance of the feature, then we will divide

each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.

4. **Calculating the Covariance of Z**

To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.

5. **Calculating the Eigen Values and Eigen Vectors**

Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.

6. **Sorting the Eigen Vectors**

In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.

7. **Calculating the new features Or Principal Components**

Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.

8. **Remove less or unimportant features from the new dataset.**

The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

Applications of Principal Component Analysis

- PCA is mainly used as the dimensionality reduction technique in various AI applications such **as computer vision, image compression, etc.**
- It can also be used for finding hidden patterns if data has high dimensions. Some fields where PCA is used are Finance, data mining, Psychology, etc.

Factor Analysis

Factor Analytics is a special technique reducing the huge number of variables into a few numbers of factors is known as factoring of the data, and managing which data is to be present in sheet comes under factor analysis. It is completely a statistical approach that is also used to describe fluctuations among the observed and correlated variables in terms of a potentially lower number of unobserved variables called *factors*.

The factor analysis technique extracts the maximum common variance from all the variables and puts them into a common score. It is a theory that is used in training the machine learning model and so it is quite related to data mining. The belief behind factor analytic techniques is that the information gained about the interdependencies between observed variables can be used later to reduce the set of variables in a dataset.

Factor analysis is a very effective tool for inspecting changeable relationships for complex concepts such as social status, economic status, dietary patterns, psychological scales, biology, psychometrics, personality theories, marketing, product management, operations research, finance, etc. It can help a researcher to investigate the concepts that are not easily measured in a much easier and quicker way directly by the cave in a large number of variables into a few easily interpretable fundamental factors.

Types of factor analysis:

1. **Exploratory factor analysis (EFA) :**

It is used to identify composite inter-relationships among items and group items that are the

part of uniting concepts. The Analyst can't make any prior assumptions about the relationships among factors. It is also used to find the fundamental structure of a huge set of variables. It lessens the large data to a much smaller set of summary variables. It is almost similar to the Confirmatory Factor Analysis(CFA).
Similarities are:

1. Evaluate the internal reliability of an amount.
2. Examine the factors represented by item sets. They presume that the factors aren't correlated.
3. Investigate the grade/class of each item.

However, some common differences, most of them are concerned about how factors are used. Basically, EFA is a data-driven approach, which allows all items to load on all the factors, while in CFA you need to specify which factors are required to load. EFA is really a nice choice if you have no idea about what common factors might exist. EFA is able to generate a huge number of possible models for your data, something which is not possible is, if a researcher has to specify factors. If you have a bit idea about what actually the models look like, and then afterwards you want to test your hypotheses about the data structure, in that case, the CFA is a better approach.

2. **Confirmatory factor analysis (CFA) :**

It is a more complex(composite) approach that tests the theory that the items are associated with specific factors. Confirmatory Factor Analysis uses a properly structured equation model to test a measurement model whereby loading on the factors allows for the evaluation of relationships between observed variables and unobserved variables.
As we know, the Structural equation modelling approaches can board measurement error easily, and these are much less restrictive than least-squares estimation thus provide more exposure to accommodate errors. Hypothesized models are tested against actual data, and the analysis would demonstrate loadings of observed variables on the latent variables (factors), as well as the correlation between the latent variables.
Confirmatory Factor Analysis allows an analyst and researcher to figure out if a relationship between a set of observed variables (also known as manifest variables) and their underlying constructs exists. It is similar to the Exploratory Factor Analysis.
The main difference between the two is:

1. Simply use Exploratory Factor Analysis to explore the pattern.
2. Use Confirmatory Factor Analysis to perform hypothesis testing.

Confirmatory Factor Analysis provides information about the standard quality of the number of factors that are required to represent the data set. Using Confirmatory Factor Analysis, you can define the total number of factors required. For example, Confirmatory Factor Analysis is able to answer questions like *Does my thousand question survey can able to measure accurately the one specific factor*. Even though it is technically applicable to any kind of discipline, it is typically used in social sciences.

3. **Multiple Factor Analysis :**

This type of Factor Analysis is used when your variables are structured in changeable groups. For example, you may have a teenager's health questionnaire with several points like sleeping patterns, wrong addictions, psychological health, mobile phone addiction, or learning disabilities.
The Multiple Factor Analysis is performed in two steps which are:-

1. Firstly, the Principal Component Analysis will perform on each and every section of the data. Further, this can give a useful eigenvalue, which is actually used to normalize the data sets for further use.
2. The newly formed data sets are going to merge into a distinctive matrix and then global PCA is performed.

4. **Generalized Procrustes Analysis (GPA) :**

The Procrustes analysis is actually a suggested way to compare then the two approximate sets of configurations and shapes, which were originally developed to equivalent to the two solutions from Factor Analysis, this technique was actually used to extend the GP Analysis so that more than two shapes could be compared in many ways. The shapes are properly aligned to achieve the target shape. Mainly GPA (Generalized Procrustes Analysis) uses geometric transformations.

Geometric progressions are :

1. Isotropic rescaling,
2. Reflection,
3. Rotation,
4. Translation of matrices to compare the sets of data.

Independent Component Analysis

(ICA) is a machine learning technique to separate independent sources from a mixed signal. Unlike principal component analysis which focuses on maximizing the variance of the data points, the independent component analysis focuses on independence, i.e. independent components.

Problem: To extract independent sources' signals from a mixed signal composed of the signals from those sources.

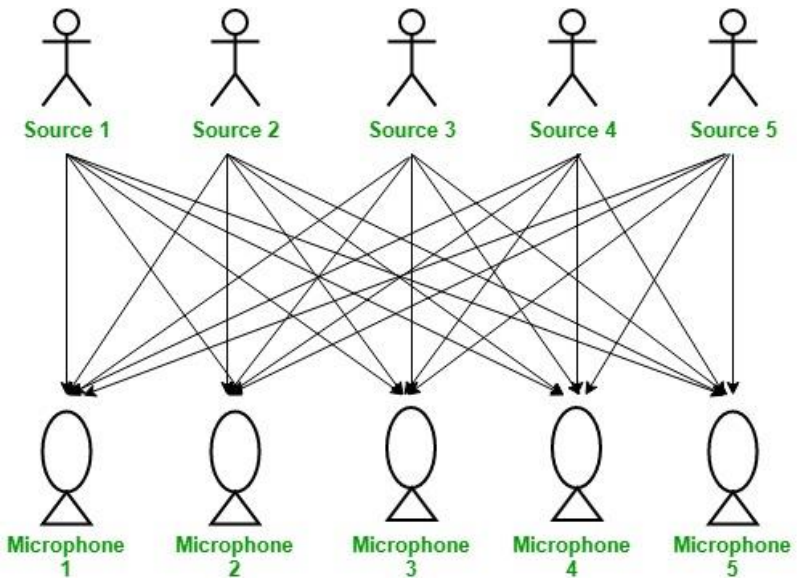
Given: Mixed signal from five different independent sources.

Aim: To decompose the mixed signal into independent sources:

- Source 1
- Source 2
- Source 3
- Source 4
- Source 5

Solution: Independent Component Analysis (ICA).

Consider *Cocktail Party Problem* or *Blind Source Separation* problem to understand the problem which is solved by independent component analysis.



Here, There is a party going into a room full of people. There is 'n' number of speakers in that room and they are speaking simultaneously at the party. In the same room, there are also 'n' number of microphones placed at different distances from the speakers which are recording 'n'

speakers' voice signals. Hence, the number of speakers is equal to the number must of microphones in the room.
Now, using these microphones' recordings, we want to separate all the 'n' speakers' voice signals in the room given each microphone recorded the voice signals coming from each speaker of different intensity due to the difference in distances between them. Decomposing the mixed signal of each microphone's recording into independent source's speech signal can be done by using the machine learning technique, independent component analysis.
[X1, X2, ..., Xn] => [Y1, Y2, ..., Yn]
where, X1, X2, …, Xn are the original signals present in the mixed signal and Y1, Y2, …, Yn are the new features and are independent components which are independent of each other

Restrictions on ICA

1. The independent components generated by the ICA are assumed to be statistically independent of each other.
2. The independent components generated by the ICA must have non-Gaussian distribution.
3. The number of independent components generated by the ICA is equal to the number of observed mixtures.

Difference between PCA and ICA –

Principal Component Analysis	Independent Component Analysis
It reduces the dimensions to avoid the problem of overfitting.	It decomposes the mixed signal into its independent sources' signals.
It deals with the Principal Components.	It deals with the Independent Components.
It focuses on maximizing the variance.	It doesn't focus on the issue of variance among the data points.
It focuses on the mutual orthogonality property of the principal components.	It doesn't focus on the mutual orthogonality of the components.
It doesn't focus on the mutual independence of the components.	It focuses on the mutual independence of the components.

<u>Locally Linear Embedding LLE)</u>

Locally Linear Embedding (LLE) is a Manifold Learning technique that is used for non-linear dimensionality reduction. It is an unsupervised learning algorithm that produces low-dimensional embeddings of high-dimensional inputs, relating each training instance to its closest neighbor.

How does LLE work?

For each training instance $x^{(i)}$, the algorithm first finds its k nearest neighbors and then tries to express $x^{(i)}$ as a linear function of them. In general, if there are m training instances in total, then

it tries to find the set of weights w which minimizes the squared distance between $x^{(i)}$ and its linear representation.

So, the cost function is given by

$$\sum_{i=1}^{m}(x^{(i)} - \sum_{j=1}^{m} w_{i,j}x^{(j)})^2$$

where $w_{i,j}$ =0, if j is not included in the k closest neighbors of i.

Also, it normalizes the weights for each training instance $x^{(i)}$,

$$\sum_{j=1}^{m} w_{i,j} = 1$$

Finally, each high-dimensional training instance $x^{(i)}$ is mapped to a low-dimensional (say, d dimensions) vector $y^{(i)}$ while preserving the neighborhood relationships. This is done by choosing d-dimensional coordinates which minimize the cost function,

$$\sum_{i=1}^{m}(y^{(i)} - \sum_{j=1}^{m} w_{i,j}y^{(j)})^2$$

Here the weights $w_{i,j}$ are kept fixed while we try to find the optimum coordinates $y^{(i)}$

Least Square optimization

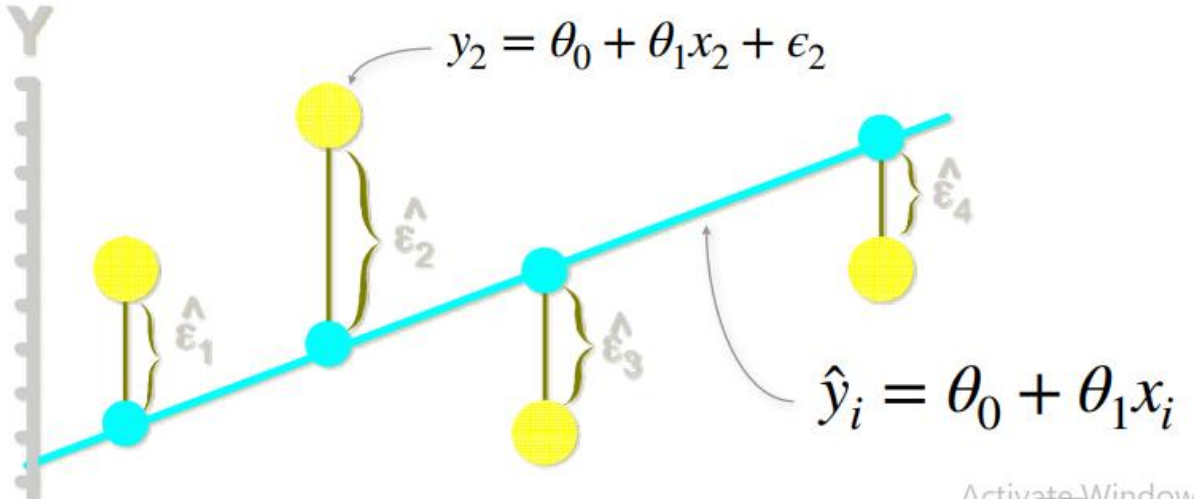
Best fit: difference between the true (observed) Y-values and the estimated Y-values is minimized:

- Positive errors offset negative errors …
- … square the error!

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}\epsilon_i^2$$

Least squares minimizes the sum of the squared errors

LS Minimizes $\sum_{i=1}^{n} \epsilon_i^2 = \epsilon_1^2 + \epsilon_2^2 + \ldots + \epsilon_n^2$



Rewrite inputs:

Each row is a feature vector paired with a label for a single input

n labeled inputs

$$X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \cdots \\ (x^{(n)})^T \end{bmatrix} \in \mathbb{R}^{n \times m}, \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \cdots \\ y^{(n)} \end{bmatrix} \in \mathbb{R}^{n}$$

m features

Rewrite optimization problem:

$$\text{minimize}_\theta \ \frac{1}{2}\|X\theta - y\|_2^2$$

*Recall $||z||_2^2 = z^T z = \sum z_i^2$

Rewrite inputs:

Each row is a feature vector paired with a label for a single input

n labeled inputs

$$X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \cdots \\ (x^{(n)})^T \end{bmatrix} \in \mathbb{R}^{n \times m}, y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \cdots \\ y^{(n)} \end{bmatrix} \in \mathbb{R}^n$$

m features

Rewrite optimization problem: $\text{minimize}_\theta \ \frac{1}{2}\|X\theta - y\|_2^2$

$$\implies \text{minimize} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \epsilon_i^2$$

*Recall $||z||_2^2 = z^T z = \sum z_i^2$

Recall the least squares optimization problem:

$$\text{minimize}_\theta \ \frac{1}{2}\|X\theta - y\|_2^2$$

What is the gradient of the optimization objective ????????

$$\nabla_\theta \frac{1}{2}\|X\theta - y\|_2^2 =$$

Chain rule:
$\nabla_\theta f(X\theta) = X^T \nabla_{X\theta} f(X\theta)$

$$X^T \nabla_{X\theta} \frac{1}{2}\|X\theta - y\|_2^2 =$$

Gradient of norm:
$\nabla_\theta \|\theta - z\|_2^2 = 2(\theta - z)$

$$\nabla_\theta \frac{1}{2}\|X\theta - y\|_2^2 = X^T(X\theta - y)$$

Recall: points where the gradient equals zero are minima.

$$\nabla_\theta \frac{1}{2}\|X\theta - y\|_2^2 = X^T(X\theta - y)$$

So where do we go from here????????

$$X^T(X\theta - y) = 0$$

Solve for model parameters θ

$$X^T X\theta - X^T y = 0 \Rightarrow X^T X\theta = X^T y$$

$$(X^T X)^{-1} X^T X\theta = (X^T X)^{-1} X^T y$$

$$\theta = (X^T X)^{-1} X^T y$$

What Is Collaborative Filtering?

Collaborative filtering is a technique that can filter out items that a user might like on the basis of reactions by similar users.

It works by searching a large group of people and finding a smaller set of users with tastes similar to a particular user. It looks at the items they like and combines them to create a ranked list of suggestions.

There are many ways to decide which users are similar and combine their choices to create a list of recommendations.

Steps Involved in Collaborative Filtering

To build a system that can automatically recommend items to users based on the preferences of other users, the first step is to find similar users or items. The second step is to predict the ratings of the items that are not yet rated by a user. So, you will need the answers to these questions:

- How do you determine which users or items are similar to one another?
- Given that you know which users are similar, how do you determine the rating that a user would give to an item based on the ratings of similar users?
- How do you measure the accuracy of the ratings you calculate?

Collaborative filtering is a family of algorithms where there are multiple ways to find similar users or items and multiple ways to calculate rating based on ratings of similar users. One important thing to keep in mind is that in an approach based purely on collaborative filtering, the similarity is not calculated using factors like the age of users, genre of the movie, or any other data about users or items. It is calculated only on the basis of the rating (explicit or implicit) a user gives to an item. For example, two users can be considered similar if they give the same ratings to ten movies despite there being a big difference in their age. The third question for how to measure the accuracy of your predictions also has multiple answers, which include error calculation techniques that can be used in many places and not just recommenders based on collaborative filtering. One of the approaches to measure the accuracy of your result is the Root Mean Square Error (RMSE), in which you predict ratings for a test dataset of user-item pairs whose rating values are already known. The difference between the known value and the predicted value would be the error. Square all the error values for the test set, find the average (or

mean), and then take the square root of that average to get the RMSE. Another metric to measure the accuracy is Mean Absolute Error (MAE), in which you find the magnitude of error by finding its absolute value and then taking the average of all error values.

Memory Based

The first category includes algorithms that are memory based, in which statistical techniques are applied to the entire dataset to calculate the predictions.

To find the rating **R** that a user **U** would give to an item **I**, the approach includes:

- Finding users similar to **U** who have rated the item **I**
- Calculating the rating **R** based the ratings of users found in the previous step

User-Based vs Item-Based Collaborative Filtering

The technique in the examples explained above, where the rating matrix is used to find similar users based on the ratings they give, is called user-based or user-user collaborative filtering. If you use the rating matrix to find similar items based on the ratings given to them by users, then the approach is called item-based or item-item collaborative filtering.

The two approaches are mathematically quite similar, but there is a conceptual difference between the two. Here's how the two compare:

- **User-based:** For a user **U**, with a set of similar users determined based on rating vectors consisting of given item ratings, the rating for an item **I**, which hasn't been rated, is found by picking out N users from the similarity list who have rated the item **I** and calculating the rating based on these N ratings.
- **Item-based:** For an item **I**, with a set of similar items determined based on rating vectors consisting of received user ratings, the rating by a user **U**, who hasn't rated it, is found by picking out N items from the similarity list that have been rated by **U** and calculating the rating based on these N ratings.

Item-based collaborative filtering was developed by Amazon. In a system where there are more users than items, item-based filtering is faster and more stable than user-based. It is effective because usually, the average rating received by an item doesn't change as quickly as the average rating given by a user to different items. It's also known to perform better than the user-based approach when the ratings matrix is sparse.

Although, the item-based approach performs poorly for datasets with browsing or entertainment related items such as MovieLens, where the recommendations it gives out seem very obvious to the target users..

Model Based

The second category covers the Model based approaches, which involve a step to reduce or compress the large but sparse user-item matrix. For understanding this step, a basic understanding of dimensionality reduction can be very helpful.

When Can Collaborative Filtering Be Used?

Collaborative filtering works around the interactions that users have with items. These interactions can help find patterns that the data about the items or users itself can't. Here are some points that can help you decide if collaborative filtering can be used:

- Collaborative filtering doesn't require features about the items or users to be known. It is suited for a set of different types of items, for example, a supermarket's inventory where items of various categories can be added. In a set of similar items such as that of a

bookstore, though, known features like writers and genres can be useful and might benefit from content-based or hybrid approaches.
- Collaborative filtering can help recommenders to not overspecialize in a user's profile and recommend items that are completely different from what they have seen before. If you want your recommender to not suggest a pair of sneakers to someone who just bought another similar pair of sneakers, then try to add collaborative filtering to your recommender spell.

Although collaborative Filtering is very commonly used in recommenders, some of the challenges that are faced while using it are the following:

- Collaborative filtering can lead to some problems like cold start for new items that are added to the list. Until someone rates them, they don't get recommended.
- Data sparsity can affect the quality of user-based recommenders and also add to the cold start problem mentioned above.
- Scaling can be a challenge for growing datasets as the complexity can become too large. Item-based recommenders are faster than user-based when the dataset is large.
- With a straightforward implementation, you might observe that the recommendations tend to be already popular, and the items from the long tail section might get ignored.

With every type of recommender algorithm having its own list of pros and cons, it's usually a hybrid recommender that comes to the rescue. The benefits of multiple algorithms working together or in a pipeline can help you set up more accurate recommenders. In fact, the solution of the winner of the Netflix prize was also a complex mix of multiple algorithms.

Collaborative Filtering Advantages & Disadvantages

Advantages

No domain knowledge necessary

We don't need domain knowledge because the embeddings are automatically learned.

Serendipity

The model can help users discover new interests. In isolation, the ML system may not know the user is interested in a given item, but the model might still recommend it because similar users are interested in that item.

Great starting point

To some extent, the system needs only the feedback matrix to train a matrix factorization model. In particular, the system doesn't need contextual features. In practice, this can be used as one of multiple candidate generators.

Disadvantages

Cannot handle fresh items

The prediction of the model for a given (user, item) pair is the dot product of the corresponding embeddings. So, if an item is not seen during training, the system can't create an embedding for it and can't query the model with this item. This issue is often called the **cold-start problem**. However, the following techniques can address the cold-start problem to some extent:

- **Projection in WALS.** Given a new item i0 not seen in training, if the system has a few interactions with users, then the system can easily compute an embedding vi0 for this item without having to retrain the whole model. The system simply has to solve the following equation or the weighted version:

$$\min_{v_{i0} \in \mathbb{R}^d} \|A_{i0} - U v_{i0}\|$$

The preceding equation corresponds to one iteration in WALS: the user embeddings are kept fixed, and the system solves for the embedding of item i0. The same can be done for a new user.

- **Heuristics to generate embeddings of fresh items.** If the system does not have interactions, the system can approximate its embedding by averaging the embeddings of items from the same category, from the same uploader (in YouTube), and so on.

Hard to include side features for query/item

Side features are any features beyond the query or item ID. For movie recommendations, the side features might include country or age. Including available side features improves the quality of the model. Although it may not be easy to include side features in WALS, a generalization of WALS makes this possible.

To generalize WALS, **augment the input matrix with features** by defining a block matrix $\bar{A}$, where:

- Block (0, 0) is the original feedback matrix A.
- Block (0, 1) is a multi-hot encoding of the user features.
- Block (1, 0) is a multi-hot encoding of the item features.

TEXT / REFERENCE BOOKS

1. Chris Albon : Machine Learning with Python Cookbook , O‟Reilly Media, Inc.2018
2. Stephen Marsland, “Machine Learning – An Algorithmic Perspective”, Second Edition, Chapman and Hall/CRC Machine Learning and Pattern Recognition Series, 2014
3. Tom M. Mitchell, Machine Learning, India Edition 2013, McGraw Hill Education
4. Machine Learning: The art and Science of algorithms that make sense of data, Peter Flach, Cambridge University Press, 2012
5. EthemAlpaydın, Introduction to machine learning, second edition, MIT press.
6. T. Hastie, R. Tibshirani and J. Friedman, “Elements of Statistical Learning”, Springer Series , 2nd edition
7. Sebastian Raschka, "Python Machine Learning",Second Edition.Packt Publication