

Machine Learning

Recommended Books

- ▶ Machine learning by Tom Mitchell
- ▶ Pattern Recognition and Machine Learning by Christopher Bishop
- ▶ Machine Learning: A Probabilistic Perspective by Kevin P. Murphy
- ▶ Hands-On Machine Learning with Scikit-Learn & TensorFlow by Aurelien Geron
- ▶ Machine Learning by Saikat Dutt, S. Chandramouli, Amit kumar Das

Few ML projects

- ▶ Giving a brain to your homemade robot
 - ▶ Make it recognize faces or learn to walk around.
- ▶ if you have tons of data of a company like, You can unearth some hidden gems
 - ▶ User logs, financial data, production data, machine sensor data, hotline stats, HR reports etc.
 - ▶ Segment customers and find the best marketing strategy for each group
 - ▶ Recommend product for each client based on what similar client bought.
 - ▶ Detect which transactions are likely to be fraudulent
 - ▶ Predict next year's revenue
 - ▶ And many more

ML journey

- ▶ When most people hear machine learning, they picture a robot
- ▶ But ML is not just a futuristic fantasy, it's already here.
- ▶ In fact, it has been around for decades in some specialized applications, such as Optical Character Recognition (OCR)
- ▶ First ML application that really became mainstream, improving the lives of hundreds of millions of people, in 1990s
 - ▶ Spam filter
- ▶ It was followed by hundreds of ML applications that now quietly power hundreds of products and features that we use regularly,
 - ▶ from better recommendations to voice search

Human Learning

- ▶ Learning is typically referred to as the process of gaining information through observation
- ▶ With more learning task can be performed more efficiently
- ▶ Types of human learning
 - ▶ Learning under expert guidance
 - ▶ Learning guided by knowledge gained from experts
 - ▶ Learning by self

Machine learning

- ▶ Is the science (and art) of programming computers so they can learn from data.
- ▶ Definitions
 - ▶ Is the field of study that gives computers the ability to learn without being explicitly programmed- Arthur Samuel, 1959
 - ▶ A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E- Tom Mitchell, 1997

Spam filter

- ▶ Example spam filter- a learning program
 - ▶ Learn to flag spam given (training set) examples of spam and regular/ham emails.
 - ▶ Each training example is called a training instance (or sample)
 - ▶ Task T- to flag spam for new emails
 - ▶ Experience E- training data
 - ▶ Performance measure P- needs to be defined, for e.g., ratio of correctly classified emails- accuracy

- Our initial approach would be to identify ‘spam’ and ‘not spam’ emails using following criteria.
 1. Email has only one image file (promotional image), It’s a SPAM
 2. Email has only link(s), It’s a SPAM
 3. Email body consist of sentence like “You won a prize money of \$ xxxxxx”, It’s a SPAM
 4. Email from our official domain “Analyticsvidhya.com” , Not a SPAM
 5. Email from known source, Not a SPAM

Why Machine Learning?

- ▶ In the early days of “intelligent” applications, many systems used handcoded rules of “if ” and “else” decisions to process data or adjust to user input.

Why use machine learning?

- ▶ Consider spam filter program using traditional programming technique
- ▶ Steps:
 - ▶ 1. what spam typically looks like?
 - ▶ Notice some words or phrases (“4U”, “credit cards”, “free” and “amazing”) tend to come up a lot in the subject
 - ▶ Similarly a few other patterns in the sender’s name, email’s body and so on
 - ▶ 2. write a detection algorithm for each of the patterns that you noticed
 - ▶ The program would flag emails as spam if a number of these patterns are detected
 - ▶ 3. test your program, and repeat steps 1 and 2 until it is good enough
- ▶ Limitations:
 - ▶ Since the problem is not trivial, the program will become a long list of complex rules
 - ▶ pretty hard to maintain

Spam filter based on machine learning techniques

- ▶ Automatically learns which words and phrases are good predictors of spam by detecting
 - ▶ unusually frequent patterns of words in the spam examples compared to the ham examples.
- ▶ The program is
 - ▶ Much shorter
 - ▶ Easier to maintain
 - ▶ And most likely more accurate

One more problem...

- ▶ If spammers keep working around the spam filter program, the programmer will need to keep writing new rules forever.
- ▶ In contrast, a spam filter based on learns what is now become unusually frequent in spam flagged by users, and ML techniques automatically it starts flagging them without the programmer's intervention.

ML is useful when

- ▶ Problem is too complex for traditional approaches or have no known algorithm
- ▶ E.g. speech recognition
 - ▶ We want a simple program capable of distinguishing the words “one” and “two”
 - ▶ One can notice that the word “two” starts with a high-pitch sound (“T”),
 - ▶ so one can hardcode an algorithm that measures high-pitch sound intensity and use that to distinguish “one” and “two”.
 - ▶ Obviously this technique will not scale to thousands of words spoken by millions of very different people in noisy environment and in dozens of languages.
 - ▶ Solution: write an algorithm that learns by itself, given many example recordings for each word.

ML can help humans learn

- ▶ ML algorithms can be inspected to see what they have learned
 - ▶ Although for some algorithms this can be tricky
 - ▶ For instance, once the spam filter has been trained on enough spam, it can easily be inspected to reveal the list of words and phrases that it believes are the best predictors of spam.
 - ▶ Sometimes this will reveal unsuspected correlations or new trends, and thereby lead to a better understanding of the problem
- ▶ Applying ML techniques to dig into large amount of data can help discover patterns that were not immediately apparent: data mining

Learning by example

- ▶ Look at some data
- ▶ Guess at a general scientific hypothesis that could explain/represent the data
- ▶ Make statements/predictions on test data based on this hypothesis

Machine Learning

Ideal Situation

- ▶ The machine is only fed with (raw) data
- ▶ And minimal (or Zero) amount of pre-built models and hypothesis
- ▶ It won't happen
- ▶ Inductive biases
 - ▶ Set of assumptions, hypothesis will always be there in some measure
 - ▶ Inductive biases are necessary since learning is an ill-posed problem
 - ▶ training data is never sufficient to find a unique solution
 - ▶ multiple hypotheses can agree with the data
- ▶ simplest consistent hypothesis is the best one

Multiple hypotheses can agree with the data

- ▶ You are given this dataset:

x	y
0	0
1	1
2	0

- ▶ You want to learn a function $h(x)$ that maps input x to output y , where $y \in \{0, 1\}$
- ▶ Multiple Hypotheses Fit the Data

Multiple Hypotheses

- ▶ Hypothesis A (piecewise rule):

$$h(x) = \begin{cases} 0 & \text{if } x=0 \text{ or } x=2 \\ 1 & \text{if } x=1 \end{cases}$$

- ▶ Hypothesis B (polynomial):

Fit a quadratic:

- ▶ $h(x) = -x^2 + 2x$

- ▶ This gives $h(0)=0, h(1)=1, h(2)=0$
 $h(0)=0, h(1)=1, h(2)=0$
 $h(0)=0, h(1)=1, h(2)=0$ – fits all 3 points.

- ▶ Hypothesis C (neural network):

- ▶ A neural net with nonlinear activations could also learn a function that outputs the same labels for these 3 inputs.

Types of ML systems

- ▶ Can be classified in broad categories based on
 - ▶ Whether or not they are trained with human supervision
 - ▶ Supervised, unsupervised, semi-supervised, sequential learning and reinforcement learning
 - ▶ Whether or not they can learn incrementally from a stream of incoming data
 - ▶ online versus batch learning
 - ▶ Whether they work by simply comparing new data points to known data points or instead detect patterns in the training data and build a predictive model
 - ▶ Instance based versus model-based learning

Supervised Learning

- ▶ classes or labels are available
- ▶ Model is built based on training data
- ▶ Model performance can be evaluated based on
 - ▶ how many mispredictions have been done based on predicted and actual values

Types of Supervised Learning problems

- ▶ Classification
 - ▶ When we are trying to predict categorical or nominal variable (no numeric values)
 - ▶ Predicting whether a tumor is malignant or benign
 - ▶ Classifying texts such as classifying a set of emails as spam or non-spam/ham
- ▶ Regression
 - ▶ When we are trying to predict a real-valued variable like (numeric)
 - ▶ Real estate, stock price, temperature, marks in an examination etc.

Types of Supervised Learning problems

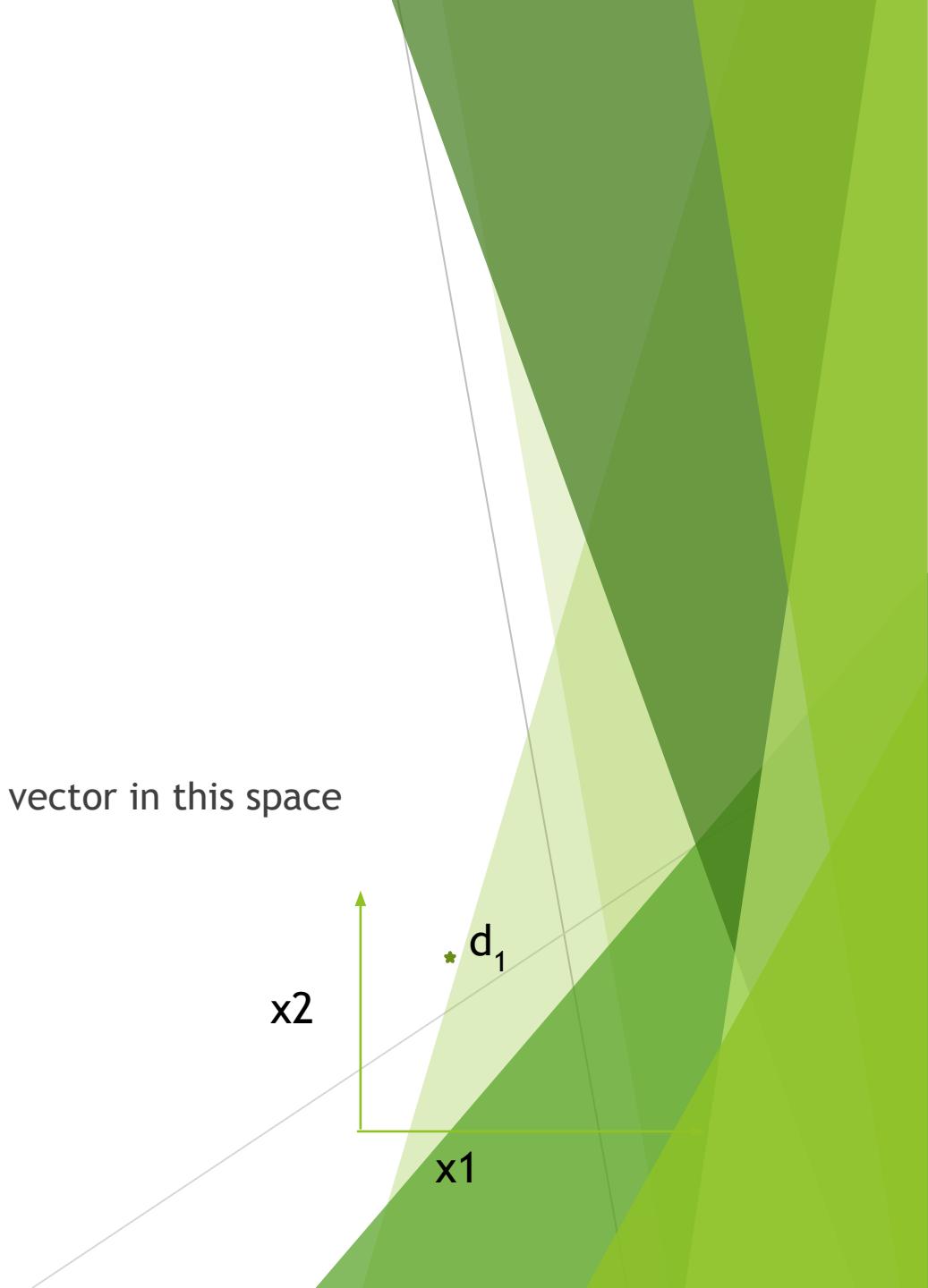
- ▶ Given examples (x, y) , we can construct/learn a prediction rule, an estimator $f: X \rightarrow Y$
 - ▶ In case of classification $f(x)$ is discrete
 - ▶ In case of regression $f(x)$ is continuous
 - ▶ In case of probability estimation $f(x)$ is the probability of x
- ▶ This is the type of inductive learning problem
 - ▶ Learning from some examples

Features

- ▶ Data objects are described by a number of *features*, that capture the basic characteristics of an object, such as the mass of a physical object or the time at which an event occurred, etc.
- ▶ Features are often called as variables, characteristics, fields, attributes, or dimensions.
- ▶ Features can be:
 - ▶ Categorical
 - ▶ Numerical
- ▶ Often we have multiple features so we have feature vector.

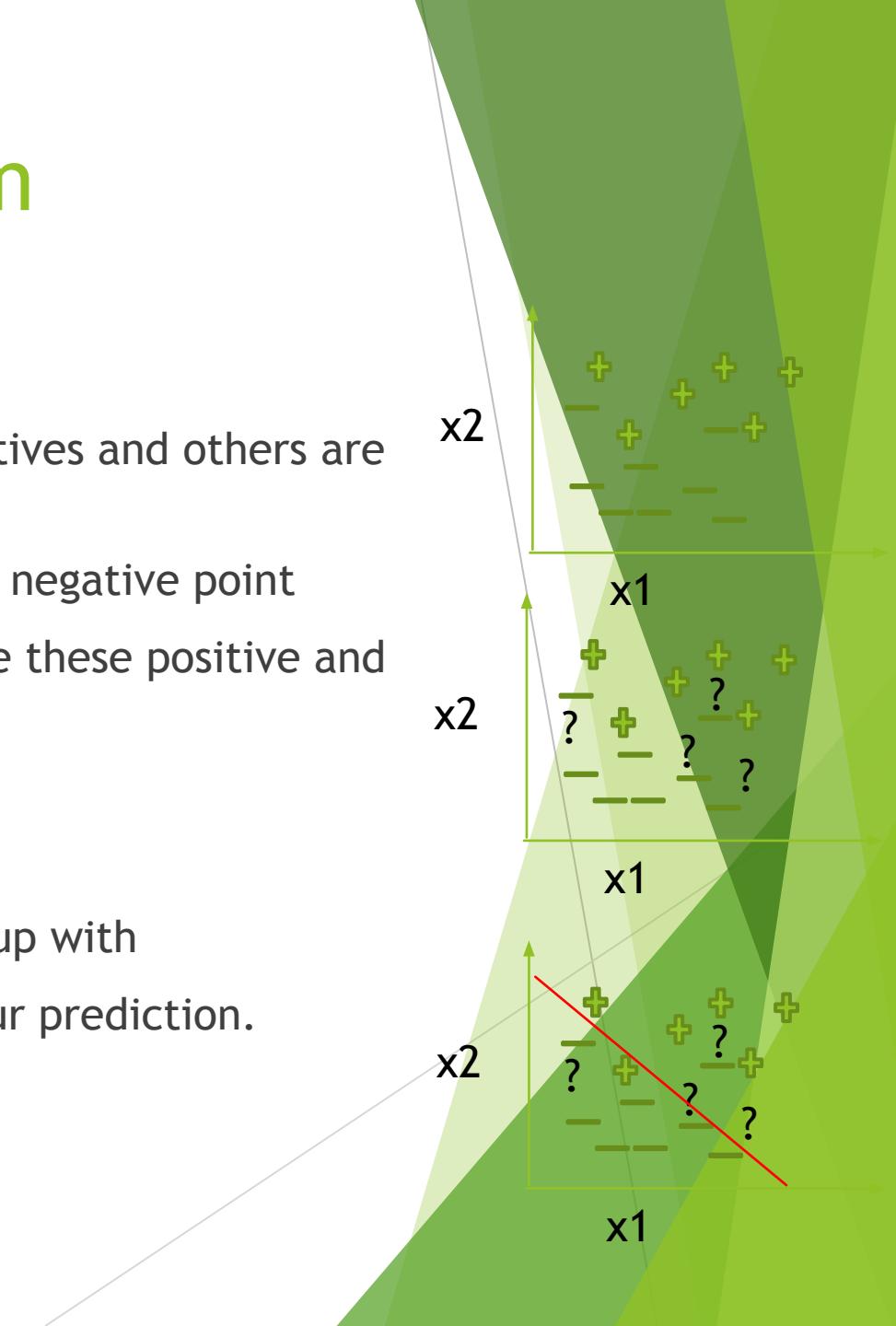
Feature space

- ▶ For n features
 - ▶ They define n-dimensional space
- ▶ For simplicity we can assume there are 2 features
 - ▶ These features define a two dimensional space
 - ▶ If we take a particular instance d_1 , lets $x_1=2$, $x_2=5$
 - ▶ So d_1 can be thought of as a point in this 2-dimensional or as a vector in this space
 - ▶ So each instance is a point in the feature space



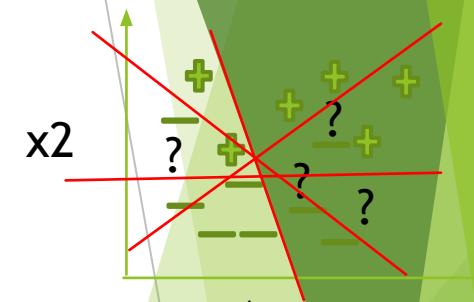
In case of classification problem

- ▶ Lets it is a two class problem
- ▶ We have some points in this feature space which are positives and others are negative.
- ▶ For a given new point, we want to know is it a positive or negative point
- ▶ So for that we want to learn a function, that will separate these positive and negative points
 - ▶ it could be a curve or a line
- ▶ So this is what inductive learning is about.
- ▶ Suppose, this red line is the function that we have come up with
- ▶ So this is the hypothesis or function that we used to do our prediction.



Hypothesis space (H)

- ▶ Instead of this particular line, we could have hypothesized other functions.
- ▶ So set of all these possible functions that we could have come up with, they define the hypothesis space
- ▶ In a particular learning problem,
 - ▶ we first define the hypothesis space that is the class of functions that you are going to consider
 - ▶ then given the data points and restrictions or biases that we impose, the algorithm try to come up with the best hypothesis.
- ▶ This is a set from which the learning algorithm will pick a hypothesis (h) that will be the output of the algorithm.
- ▶ So we can think supervised learning as a device which explores (searches) the hypothesis space in order to find out one of the hypothesis which satisfies certain criteria.



Function representation

- ▶ A function is represented in terms of features.
- ▶ In order to describe a function, we need two things to decide
 - ▶ Features of the vocabulary
 - ▶ Function class or the type of the function or the language of the function
- ▶ So based on the features and the language we can define our hypothesis space.
- ▶ To make predictions various types of representations are considered
 - ▶ Linear function to act as a discriminator
 - ▶ Decision tree
 - ▶ Multivariate linear function
 - ▶ Single layer perceptron
 - ▶ Multi-layer neural networks

Terminology

- ▶ Instance (example) - (x, y)
- ▶ Training data - set of examples
- ▶ Instance space (X)
 - ▶ Which describe all possible instances
 - ▶ set of all possible objects that can be described by the features
- ▶ Concept (c)
 - ▶ We are trying to learn a concept
 - ▶ In a classification problem where we want to learn a particular class (in case of two class classification problem positive classes we can think of the positive examples (that we want to learn) as the concept)
 - ▶ so out of all possible instances in the instance space, positive examples belong to the concept.
 - ▶ $c \in X$
 - ▶ c is unknown and what we are trying to find.

For classification problem

- ▶ Input: training set $S \subseteq X$
- ▶ Output: a hypothesis $h \subseteq H$

Size of the Hypothesis Space

- ▶ If 4 Boolean features
 - ▶ 2^4 total possible instance
 - ▶ 2^{16} total Boolean functions
- ▶ The Hypothesis space is very large and it is not possible to look every hypothesis individually to select the best hypothesis that we want
- ▶ So we put some restrictions on the hypothesis space
 - ▶ So you select a hypothesis language
- ▶ This hypothesis language may be an unrestricted hypothesis language, for e.g. all possible Boolean functions or may be a restricted language
- ▶ If you restrict hypothesis language, it reflects an inductive bias of the learner.

Unrestricted hypothesis language

- ▶ Decision Trees (with high depth):
 - ▶ While decision trees can be restricted by depth, allowing them to be relatively simple, they can also be trained with high depth, allowing them to fit complex, non-linear patterns in the data.
- ▶ Neural Networks (with many layers and nodes):
 - ▶ Neural networks, particularly deep neural networks, have a very large hypothesis space and can learn highly complex relationships within data.

Inductive Bias

- ▶ When we choose a hypothesis space we need to make some assumptions:
- ▶ Two types of assumptions that we can make/bias
 - ▶ Put Restrictions on the type of functions
 - ▶ Example instead of considering all the Boolean formulas we will consider only conjunctive Boolean formulas
 - ▶ For regression problem, we can say we are looking at linear functions or we can look at 4th degree polynomials, or n-degree polynomials or any polynomials
 - ▶ Preference
 - ▶ Given a particular language that you have chosen you say that
 - ▶ I am considering all the possible polynomials but I will prefer polynomials of lower degree
 - ▶ I am considering all possible Boolean functions but I will prefer Boolean functions which can be described in small size.

Most Important Supervised Learning Algorithms

- ▶ Naïve Bayes
- ▶ K-nearest Neighbors
- ▶ Linear Regression
- ▶ Logistic Regression
- ▶ Support Vector Machine (SVMs)
- ▶ Decision Trees and Random Forests
- ▶ Neural networks

Practical Applications

- ▶ Handwriting recognition
- ▶ Stock market prediction
- ▶ Disease prediction
- ▶ Fraud detection, etc.

Unsupervised learning

- ▶ Is used when there is no idea about the class or label of a particular data.
- ▶ The model has to find patterns in the data.
- ▶ Unlabeled dataset is given to the model and Input is grouped
- ▶ Difficult to measure whether the model did something useful or interesting
 - ▶ Homogeneity of records grouped together is the only measure.

Types of Unsupervised Learning problems

- ▶ Clustering
- ▶ Association
- ▶ representation learning
 - ▶ How to represent data points in a best possible way so that the task (whatever in our hand) can become easier.
 - ▶ example we may learn a better way to represent an image in place of a bunch of pixel values.
- ▶ Dimensionality reduction
- ▶ Unsupervised density estimation

Most Important unsupervised Learning Algorithms

- ▶ Clustering
 - ▶ K-means
 - ▶ Hierarchical Cluster Analysis (HCA)
 - ▶ Expectation maximization
- ▶ Visualization and dimensionality reduction
 - ▶ Principal Component Analysis (PCA)
 - ▶ Kernel PCA
 - ▶ Locally-Linear Embedding (LLE)
 - ▶ T-distributed Stochastic Neighbor Embedding (t-SNE)
- ▶ Association rule learning
 - ▶ Apriori
 - ▶ Eclat

Practical Applications

- ▶ Market basket analysis
- ▶ Recommender system
- ▶ Customer segmentation, etc.

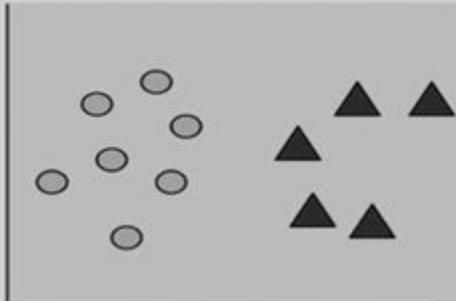
Machine Learning

Semi Supervised Learning

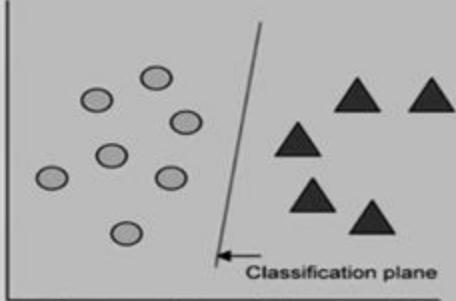
Semi Supervised Learning

- It is a combination of supervised and unsupervised learning
- It is used when we have labelled training data and a larger amount of unlabelled training data
- We try to come up with some learning algorithm that can work even when the training data is limited

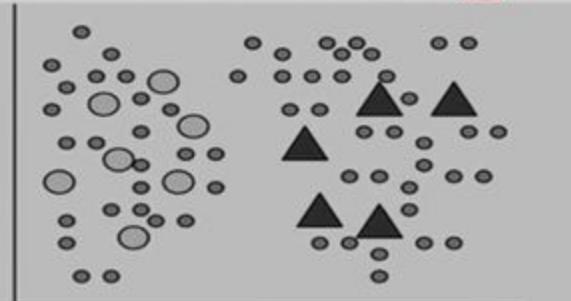
Semi-supervised learning



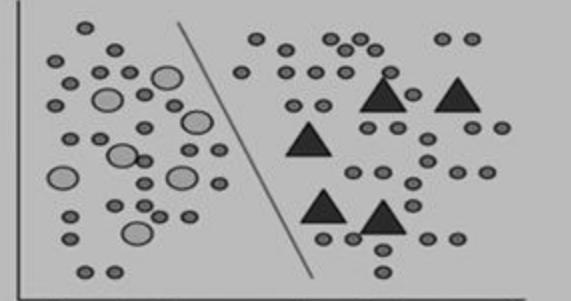
Labeled Data
(a)



Supervised Learning
(c)



Labeled and Unlabeled Data
(b)



Semi-Supervised Learning
(d)

Sequential learning

Sequential learning

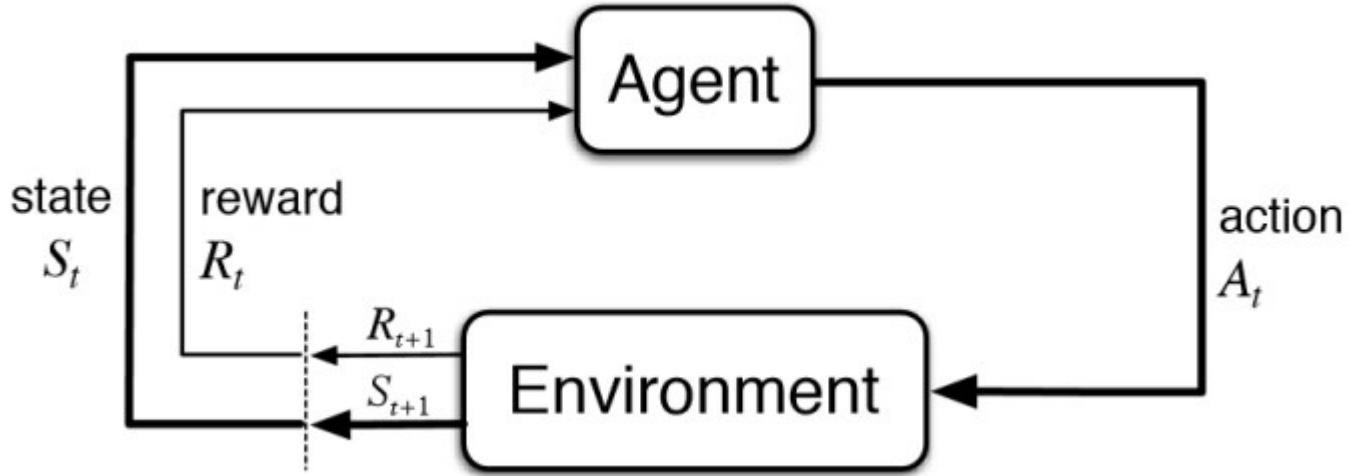
- Learning not in a single shot but over the time
- It predicts and then gets feedback and based on that it learns.

Sequential learning

- Online learning
 - Stock market prediction
 - Based on few experts suggestions you buy some stock and then based on actual feedback you learn over the time. (not only you but all experts also learns based on feedback: this is called full learning)
- Multi-armed Bandits
 - Based on the symptoms of a patient your algorithm suggest for treatments and you go for suppose 2nd treatment and then based on feedback (the patient recovers or deteriorates) you learn but you don't know the output of other treatments (so not full learning)
- Reinforcement learning
 - Robot navigation
 - Robot need to reach from one part of a room to other end of the room. It can take decisions- left, right, back, forward based on the surrounding place. So mapping is required to learn from surrounding places to decisions. Based on taken decisions it will learn.

Reinforcement Learning

- ▶ No labels, No prepared training data
- ▶ These are neither supervised nor unsupervised in nature
- ▶ In RL, an agent learns via its interactions with an environment (feedback-driven policy learning)
- ▶ Learning by trial and error
- ▶ Data are not given. The learning agent must generate them by interacting with the environment
- ▶ Typically these are problems where you are learning to control the behavior of the system
- ▶ The model learns and updates itself through reward/punishment
- ▶ It learns by itself what is the best strategy, called a policy, to get the most reward over time.
- ▶ Model is evaluated by means of the reward function after it had some time to learn.



- ▶ We have an agent which acts in an environment
- ▶ So the agent is in a particular state and can take action and this action can impact the environment
- ▶ and environment goes to a new state and give reward/penalty/neutral to agent
- ▶ Agent is continually acting in this environment
- ▶ And the reinforcement learner learn a policy- given a state what action to take so that not only the short term reward is optimized but the overall utility of the agent over its entire time horizon is optimized.

Types of Reinforcement Learning Problems

- ▶ can be used for a variety of **planning problems** including
 - ▶ travel plans,
 - ▶ budget planning and
 - ▶ business strategy.

Most Important Reinforcement Learning Algorithms

- ▶ Q-learning
- ▶ SARSA (State–action–reward–state–action) is an algorithm for learning a Markov decision process policy

Practical Applications

- ▶ Self-driving cars
- ▶ Intelligent robots
- ▶ AlphaGo Zero (the latest version of DeepMind's AI system playing Go)

Batch learning (offline learning)

- ▶ The system is incapable of learning incrementally
- ▶ It must be trained using all the available data
- ▶ This will take a lot of time and computing resources, so it is typically done offline.
- ▶ First the system is trained, then it is launched into production without learning anymore

Online (Incremental) Learning

System is trained incrementally by feeding it data instances sequentially, either individually or by small groups called mini batches.

Each learning step is fast and cheap, so the system can learn about new data on the fly, as it arrives

Useful for systems that receive data as continuous flow (e.g., stock prices) and need to adapt to change rapidly.

It is also a good option if there is limited computing resources.

We can discard the data on which it has learned.

Can be used to train systems on huge datasets that cannot fit in one machine's main memory (out-of-core learning)

Whole process is usually done offline (not on the live system)

Online (Incremental) Learning

- ▶ Important parameter- learning rate
- ▶ If too high
 - ▶ System will adapt to new data but will forget old data
- ▶ If too slow
 - ▶ It will learn more slowly

Machine Learning Tasks

Task	Performance Measure
Classification	error
Regression	error
Clustering	scatter of points in a single cluster/purity
Associations	support/confidence
Reinforcement learning	cost/reward

Inductive learning versus Deductive Learning

- ▶ Deductive Learning
 - ▶ logical,
 - ▶ From facts
- ▶ Inductive learning
 - ▶ From evidence
 - ▶ Empirical suggestions

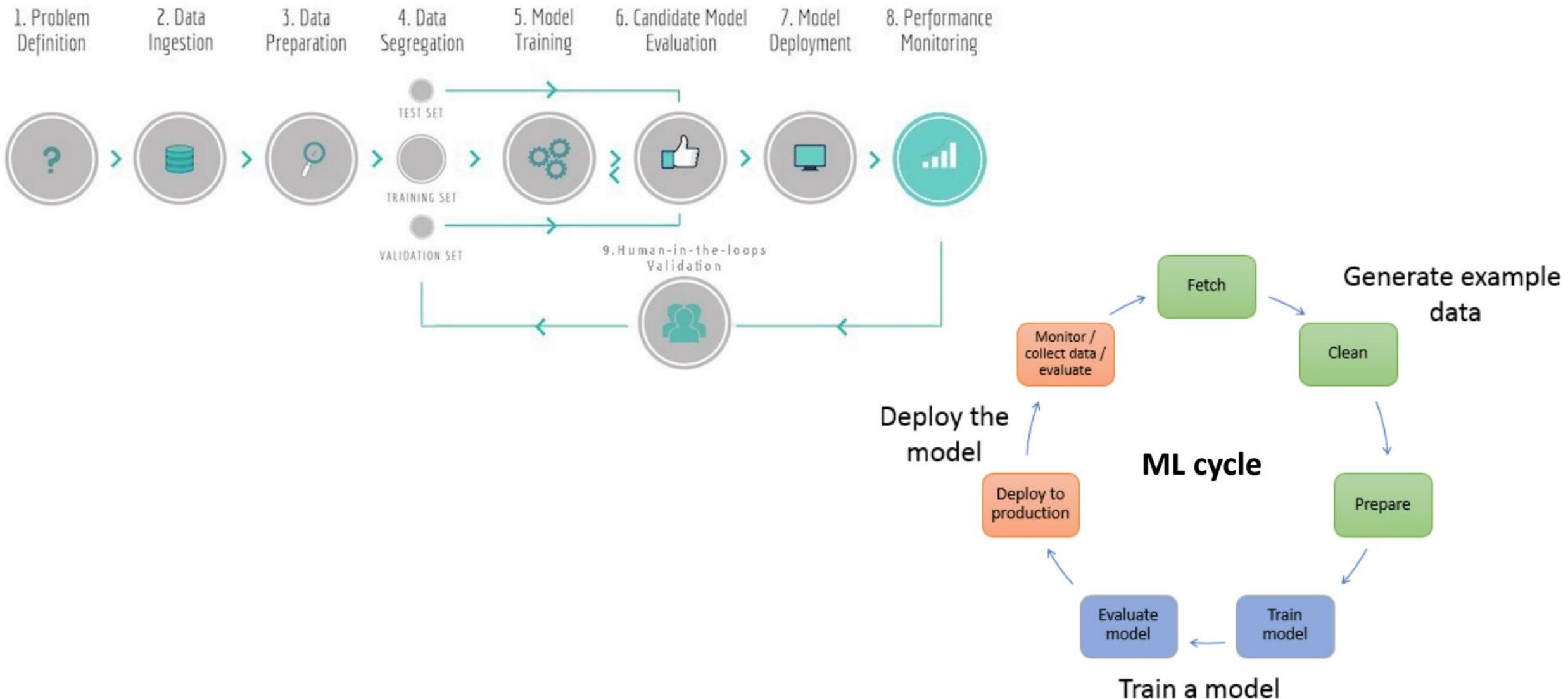
Main Challenges of Machine learning

- ▶ How good a model is?
- ▶ How do I choose a model?
- ▶ Do I have enough data?
- ▶ Is the data of sufficient quality?
 - ▶ Errors in data
 - ▶ Missing values
- ▶ How confident can I be of the results?
- ▶ Am I describing the data correctly?
 - ▶ Are age and income enough? Should I look at gender also?
 - ▶ How should I represent age? As a number, or as young, middle age, old etc.
- ▶ How to ensure generalization
 - ▶ Ability of an ML algorithm to do well on future test data

Main Challenges of Machine learning

- ▶ Examples of bad data
 - ▶ Insufficient quantity of training data
 - ▶ Nonrepresentative training data
 - ▶ Poor quality data
 - ▶ Irrelevant features
- ▶ Examples of bad algorithms
 - ▶ Overfitting the training data
 - ▶ Underfitting the training data

ML pipeline / workflow / cycle in the production process



Assumptions in ML

- •IID (Independent and Identically distribution of data) assumptions
 - Independent
 - Each data sample is drawn independently of the others, meaning the outcome of one sample does not affect the outcome of another
 - This ensures there are no dependencies or correlations between the samples in the dataset
 - Identically
 - All data points come from the same underlying distribution
 - if the data is drawn from a Gaussian distribution, all samples must follow the same Gaussian distribution
 - This assumption ensures the learning algorithm can generalize from the training data to unseen data, as the test set is assumed to follow the same distribution
 - Test and training data are generated by a probability distribution

performance of machine learning

- ▶ Make the training error small
- ▶ reduce the gap between training and test error

Testing and validating

- ▶ We split data into two sets: training set and test set
- ▶ The error rate on new cases is called the generalization error (or out-of-sample error)
- ▶ By evaluating the model on test set , we get an estimation of generalization error.
- ▶ If the training error is low but the generalization error is high, it means the model is overfitting the training data.

Underfitting the training data

- ▶ Opposite of overfitting
- ▶ It occurs when model is too simple to learn the underlying structure of the data.
- ▶ If your model is underfitting the training data, adding more training examples will not help.
- ▶ The main options to fix this problem are:
 - ▶ Selecting a more powerful model, with more parameters
 - ▶ Feeding better features to the learning algorithm (feature engineering)
 - ▶ Reducing the constraints on the model
- ▶ In supervised learning, **underfitting** happens when a model is too simple to represent all the relevant class characteristics i.e. unable to capture the underlying pattern of the data.
 - ▶ These models usually have high bias and low variance.
 - ▶ High training error and high test error.
- ▶ It happens when we have very less amount of data to build an accurate model or when we try to build a linear model with a nonlinear data.
- ▶ Also, these kind of models are very simple to capture the complex patterns in data like Linear and logistic regression.

Overfitting the training data

- ▶ The gap between training and test error is too large
- ▶ Model performs well on training data, but it does not generalize well.
- ▶ Happens when model is too complex relative to the amount and noisiness of the training data.
- ▶ Possible solutions are:
 - ▶ To simplify the model by selecting one with fewer parameters (e.g.. A linear model rather than a high-degree polynomial model), by reducing the number of attributes in the training data or by constraining the model
 - ▶ To gather more training data
 - ▶ To reduce the noise in the training data(e.g., fix data errors and remove outliers)
- ▶ Constraining a model to make it simpler and reduce the risk of overfitting is called regularization.
- ▶ In supervised learning, **overfitting** happens when our model is too complex and fits irrelevant characteristics (noise) along with the underlying pattern in data.
- ▶ It happens when we train our model a lot over noisy dataset.
- ▶ These models have low bias and high variance.
- ▶ Low training error and high test error
- ▶ These models are very complex like Decision trees which are prone to overfitting.

Understanding the Bias-Variance Tradeoff

- ▶ prediction errors
 - ▶ bias and variance
- ▶ There is a tradeoff between a model's ability to minimize bias and variance
- ▶ Understanding of these errors avoids the mistake of overfitting and underfitting
- ▶ If models are more flexible (less bias), it becomes more data sensitive (high variance)
- ▶ If models are less data sensitive (less variance), it becomes less flexible (high bias)

Bias

- ▶ Also called “error due to squared bias”
- ▶ difference between the average prediction of our model and the correct value which we are trying to predict
- ▶ Model with high bias pays very little attention to the training data and oversimplifies the model
- ▶ It always leads to high error on training and test data
- ▶ A high level of bias can lead to underfitting
 - ▶ which occurs when the algorithm is unable to capture relevant relations between features and target outputs
- ▶ A linear algorithm often has high bias, which makes them learn fast.
- ▶ In contrast, nonlinear algorithms often have low bias.

variance

- ▶ Variance indicates how much the estimate of the target function will alter if different training data were used.
- ▶ Variance is the variability of model prediction for a given data point or a value which tells us spread of our data.
- ▶ Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before.
- ▶ As a result, such models perform very well on training data but has high error rates on test data.
- ▶ Variance can lead to overfitting, in which small fluctuations in the training set are magnified.

Bias-Variance Summary

- ▶ variance refers to the amount by which that would change if we estimated it using a different training data set
 - ▶ more flexible models have higher variance
- ▶ bias refers to the error that is introduced by approximating a extremely complex problem by a much simpler model
 - ▶ more flexible models have low bias

UNDERSTANDING DATASETS

NEED OF UNDERSTANDING DATASET

- to understand the incoming data and create basic understanding about the nature and quality of the data
- helps to select and then how to apply the model
- The first step in machine learning activity starts with data
- In case of supervised learning, it is the labelled training data set followed by test data which is not labelled
- In case of unsupervised learning, there is no question of labelled data but the task is to find patterns in the input data
- A thorough review and exploration of the data is needed to understand the type of the data, the quality of the data and relationship between the different data elements.

NEED OF UNDERSTANDING DATASET

- Based on that, multiple pre-processing activities may need to be done on the input data before we can go ahead with core machine learning activities
- Following are the typical **preparation** activities done once the input data comes into the machine learning system:
 - Understand the type of data in the given input data set.
 - Explore the data to understand the nature and quality.
 - Explore the relationships amongst the data elements, e.g. inter-feature relationship.
 - Find potential issues in data.
 - Do the necessary remediation, e.g. impute missing data values, etc., if needed.
 - Apply pre-processing steps, as necessary.

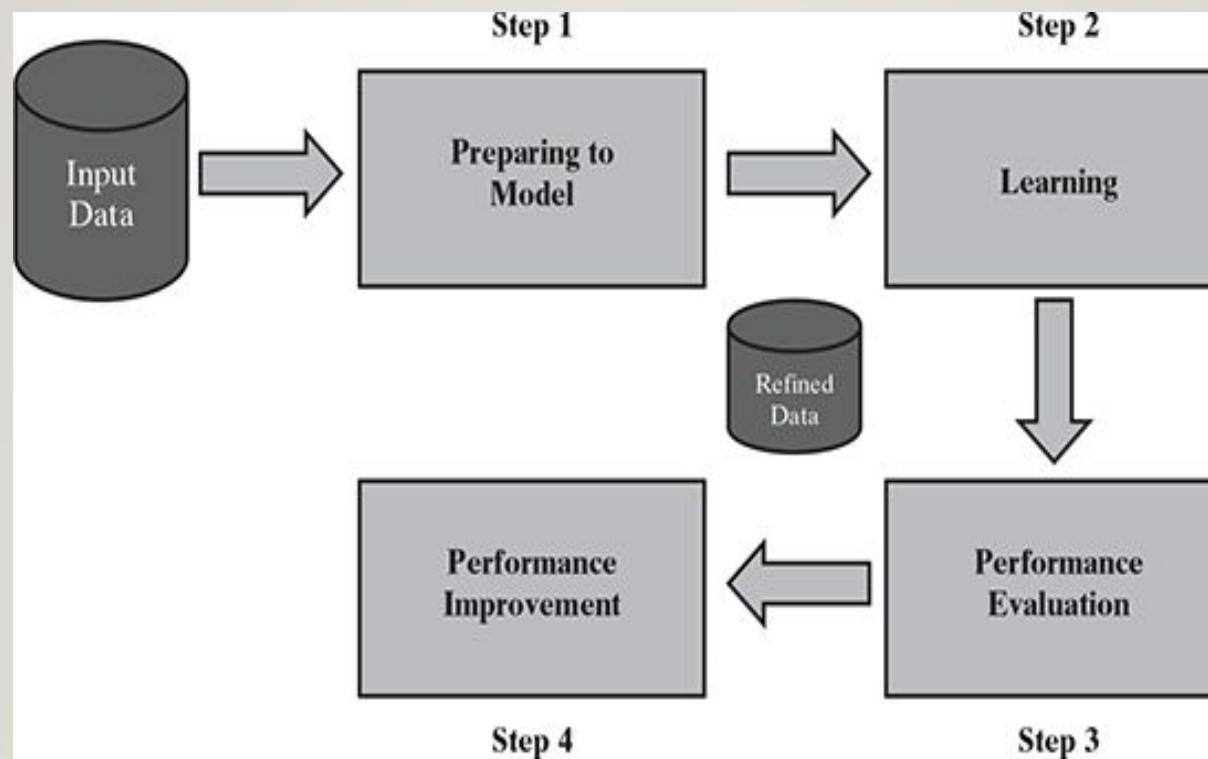
NEED OF UNDERSTANDING DATASET

- Once the data is prepared for modelling, then the learning tasks start off. As a part of it, do the following activities:
 - The input data is first divided into parts – the training data and the test data (called holdout). This step is applicable for supervised learning only.
 - Consider different models or learning algorithms for selection.
 - Train the model based on the training data for supervised learning problem and apply to unknown data. Directly apply the chosen unsupervised model on the input data for unsupervised learning problem.

NEED OF UNDERSTANDING DATASET

- After the model is selected, trained (for supervised learning), and applied on input data, the performance of the model is evaluated.
- Based on options available, specific actions can be taken to improve the performance of the model, if possible.

DETAILED PROCESS OF MACHINE LEARNING



BASIC TYPES OF DATA IN MACHINE LEARNING

- A data set is a collection of related information or records.
- Examples
 - data set on students in which each record consists of information about a specific student.
 - data set on student performance which has records providing performance, i.e. marks on the individual subjects.
- Each row of a data set is called a record.
- Each data set also has multiple attributes, each of which gives information on a specific characteristic.
- Attributes can also be termed as feature, variable, dimension or field.

EXAMPLES OF DATA SET HAVING FOUR DIMENSIONAL DATA SPACE

Student data set:

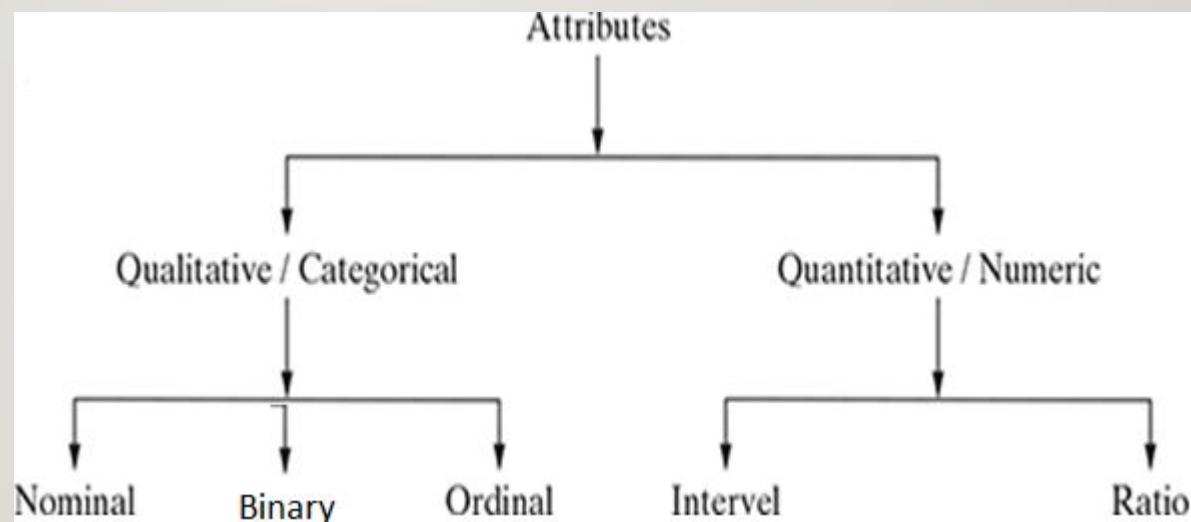
Roll Number	Name	Gender	Age
129/011	Mihir Karmarkar	M	14
129/012	Geeta Iyer	F	15
129/013	Chanda Bose	F	14
129/014	Sreenu Subramanian	M	14
129/015	Pallav Gupta	M	16
129/016	Gajanan Sharma	M	15

Student performance data set:

Roll Number	Maths	Science	Percentage
129/011	89	45	89.33%
129/012	89	47	90.67%
129/013	68	29	64.67%
129/014	83	38	80.67%
129/015	57	23	53.33%
129/016	78	35	75.33%

DATA TYPES

- Data can broadly be divided into following two types:
 - Qualitative data
 - Quantitative data



QUALITATIVE DATA

- Qualitative data is also called **categorical data**
- Qualitative data can be further subdivided into two types as follows:
 - Nominal data
 - Binary data
 - Ordinal data

NOMINAL DATA

- **Nominal data** is one which has no numeric value, but a named value.
- Nominal values cannot be quantified.
- Examples of nominal data are
 - Blood group: A, B, O, AB, etc.
 - Nationality: Indian, American, British, etc.
 - Gender: Male, Female, Other
- A special case of nominal data is when only two labels are possible, e.g. pass/fail as a result of an examination.
 - This sub-type of nominal data is called 'dichotomous'.

NOMINAL DATA

- mathematical operations such as addition, subtraction, multiplication, etc. cannot be performed on nominal data.
- statistical functions such as mean, variance, etc. can also not be applied on nominal data.
- However, a basic count is possible.
 - So mode, i.e. most frequently occurring value, can be identified for nominal data.

BINARY DATA

- **A binary attribute is a nominal attribute with only two categories or states: 0 or 1,**
 - where 0 typically means that the attribute is absent, and 1 means that it is present.
 - Example- Smoker describing a patient who smokes (1) or who does not smoke (0)
- **Binary attributes are referred to as Boolean if the two states correspond to true and false.**
- **A binary attribute is symmetric**
 - if both of its states are equally valuable and carry the same weight; that is, there is no preference on which outcome should be coded as 0 or 1.
 - One such example could be the attribute gender having the states male and female.
- **A binary attribute is asymmetric**
 - if the outcomes of the states are not equally important, such as the positive and negative outcomes of a medical test for HIV.
 - By convention, we code the most important outcome, which is usually the rarest one, by 1 (e.g., HIV positive) and the other by 0 (e.g., HIV negative)

ORDINAL DATA

- **Ordinal data** in addition to possessing the properties of nominal data, can also be naturally ordered.
- they can be arranged in a sequence of increasing or decreasing value so that we can say whether a value is better than or greater than another value.
- Examples of ordinal data are
 - Customer satisfaction: ‘Very Happy’, ‘Happy’, ‘Unhappy’, etc.
 - Grades: A, B, C, etc.
 - Hardness of Metal: ‘Very Hard’, ‘Hard’, ‘Soft’, etc.

ORDINAL DATA

- Like nominal data, basic counting is possible for ordinal data.
- Hence, the mode can be identified.
- Since ordering is possible in case of ordinal data, median, and quartiles can be identified in addition.
- Mean can still not be calculated.

QUANTITATIVE DATA

- **Quantitative data** relates to information about the quantity of an object – hence it can be measured.
- For example, if we consider the attribute ‘marks’, it can be measured using a scale of measurement.
- Quantitative data is also termed as numeric data.
- There are two types of quantitative data:
 - Interval data
 - Ratio data

INTERVAL-SCALED DATA

- Interval data is numeric data for which not only the order is known, but the exact difference between values is also known.
- An ideal example of interval data is Celsius temperature.
- The difference between each value remains the same in Celsius temperature.
- For example, the difference between 12°C and 18°C degrees is measurable and is 6°C as in the case of difference between 15.5°C and 21.5°C .
- Other examples include date, time, etc.

INTERVAL-SCALED DATA

- For interval data, mathematical operations such as addition and subtraction are possible.
- For that reason, for interval data, the central tendency can be measured by mean, median, or mode.
- Standard deviation can also be calculated.

INTERVAL-SCALED DATA

- However, interval data do not have something called a ‘true zero’ value.
- For example, there is nothing called ‘0 temperature’ or ‘no temperature’.
- Hence, only addition and subtraction applies for interval data.
- The ratio cannot be applied.
- This means, we can say a temperature of 40°C is equal to the temperature of $20^{\circ}\text{C} +$ temperature of 20°C .
- However, we cannot say the temperature of 40°C means it is twice as hot as in temperature of 20°C .

RATIO-SCALED DATA

- **Ratio data** represents numeric data for which exact value can be measured.
- Absolute zero is available for ratio data.
- Also, these variables can be added, subtracted, multiplied, or divided.
- The central tendency can be measured by mean, median, or mode and methods of dispersion such as standard deviation.
- Examples of ratio data include height, weight, age, salary, etc.

ANOTHER WAY TO CATEGORIZE DATA

- attributes can also be categorized into types based on a number of values that can be assigned.
- The attributes can be either discrete or continuous based on this factor

DISCRETE ATTRIBUTE

- It has a finite or countably infinite set of values, which may or may not be represented as integers.
- Examples: hair color, smoker, medical test, and drink size each have a finite number of values, and so are discrete.
- Note: discrete attributes may have numeric values, such as 0 and 1 for binary attributes or, the values 0 to 110 for the attribute age.
- An attribute is countably infinite if
 - the set of possible values is infinite but the values can be put in a one-to-one correspondence with natural numbers.
 - For example, the attribute customer ID is countably infinite. The number of customers can grow to infinity, but in reality, the actual set of values is countable (where the values can be put in one-to-one correspondence with the set of integers).
 - Zip codes are another example.

CONTINUOUS ATTRIBUTE

- If an attribute is not discrete, it is continuous.
- continuous values are real numbers.
- Continuous attributes are typically represented as floating-point variables.

ANOTHER WAY TO CATEGORIZE DATA

- Discrete attributes can assume a finite or countably infinite number of values.
 - Nominal attributes such as roll number, street number, pin code, etc. can have a finite number of values
 - whereas numeric attributes such as count, rank of students, etc. can have countably infinite values.
- A special type of discrete attribute which can assume two values only is called binary attribute.
- Examples of binary attribute include male/ female, positive/negative, yes/no, etc.

ANOTHER WAY TO CATEGORIZE DATA

- Continuous attributes can assume any possible value which is a real number.
- Examples of continuous attribute include length, height, weight, price, etc.

AUTO MPG DATA SET AVAILABLE IN THE UCI REPOSITORY

- the attributes such as ‘mpg’, ‘cylinders’, ‘displacement’, ‘horsepower’, ‘weight’, ‘acceleration’, ‘model year’, and ‘origin’ are all numeric.

- Out of these attributes, ‘cylinders’, ‘model year’, and ‘origin’ are discrete in nature as the only finite number of values can be assumed by these attributes.
- The remaining of the numeric attributes, i.e. ‘mpg’, ‘displacement’, ‘horsepower’, ‘weight’, and ‘acceleration’ can assume any real value. Hence, these attributes are continuous in nature.
- The only remaining attribute ‘car name’ is of type categorical, or more specifically nominal.
- This data set is regarding prediction of fuel consumption in miles per gallon, i.e. the numeric attribute ‘mpg’ is the target attribute.

mpg	cylinder	displace- ment	horse- power	weight	accel- eration	model year	origin	car name
18	8	307	130	3504	12	70	1	Chevrolet chev- elle malibu
15	8	350	165	3693	11.5	70	1	Buick skylark 320
18	8	318	150	3436	11	70	1	Plymouth satellite
16	8	304	150	3433	12	70	1	Amc rebel sst
17	8	302	140	3449	10.5	70	1	Ford torino
15	8	429	198	4341	10	70	1	Ford galaxie 500
14	8	454	220	4354	9	70	1	Chevrolet impala
14	8	440	215	4312	8.5	70	1	Plymouth fury iii
14	8	455	225	4425	10	70	1	Pontiac catalina
15	8	390	190	3850	8.5	70	1	Amc acbassador dpl
15	8	383	170	3563	10	70	1	Dodge challenger se
14	8	340	160	3609	8	70	1	Plymouth ' cuda 340
15	8	400	150	3761	9.5	70	1	Chevrolet monte carlo
14	8	455	225	3086	10	70	1	Buick estate wagon (sw)
24	4	113	95	2372	15	70	3	Toyota corona mark ii
22	6	198	95	2933	15.5	70	1	Plymouth duster
18	6	199	97	2774	15.5	70	1	Amc hornet

AUTO MPG DATA SET AVAILABLE IN THE UCI REPOSITORY: A SNAPSHOT OF THE FIRST FEW ROWS OF THE DATA SET.

Options Available

- Since the attributes ‘cylinders’ or ‘origin’ have a small number of possible values, one may prefer to treat it as a categorical or qualitative attribute and explore in that way.

BASIC STATISTICAL DESCRIPTION OF DATA

- Measuring the central tendency
 - Mean, Median, Mode
- Measuring the Dispersion of Data
 - Range, Quartiles, Variance, Standard Deviation, and Interquartile Range
- Five-Number Summary, Boxplots, and Outliers

BASIC STATISTICAL DESCRIPTION OF DATA

- Graphic Displays of Basic Statistical Descriptions of Data
 - Quantile Plot, Quantile–Quantile Plot, Histograms, Scatter Plots and Data Correlation

UNDERSTANDING CENTRAL TENDENCY

- To understand the nature of numeric variables, we can apply the measures of central tendency of data, i.e.
 - mean and median.
- In statistics, measures of central tendency help us understand the central point of a set of data.
- Mean, by definition, is a sum of all data values divided by the count of data elements.

UNDERSTANDING CENTRAL TENDENCY

- Median, on contrary, is the value of the element appearing in the middle of an ordered list of data elements.
- If we consider the 5 data elements, the ordered list would be – 21, 34, 67, 89, and 96.
- Since there are 5 data elements, the 3rd element in the ordered list is considered as the median.
 - Hence, the median value of this set of data is 67.

UNDERSTANDING CENTRAL TENDENCY

- why two measures of central tendency
 - Mean is likely to get shifted drastically even due to the presence of a small number of outliers.
 - If we observe that for certain attributes the deviation between values of mean and median are quite high, we should investigate those attributes

UNDERSTANDING DATA SPREAD

- If a large deviation between mean and median
 - To drill down more, we need to look at the entire range of values of the attributes,
 - may be too vast to review manually. So we will take a granular view of the data spread in the form of
 - Dispersion of data
 - Position of the different data values
- Dispersion of data
 - attribute values are quite concentrated around the mean
 - attribute values are extremely spread out

-
- To measure the extent of dispersion of a data
 - the variance of the data is measured.
 - Larger value of variance or standard deviation indicates more dispersion in the data and vice versa.

MEASURING DATA VALUE POSITION

- When the data values of an attribute are arranged in an increasing order,
 - median gives the central data value, which divides the entire data set into two halves.
 - Similarly, if the first half of the data is divided into two halves so that each half consists of one-quarter of the data set, then that median of the first half is known as first quartile or Q_1 .
 - In the same way, if the second half of the data is divided into two halves, then that median of the second half is known as third quartile or Q_3 .
 - The overall median is also known as second quartile or Q_2 .
 - So, any data set has five values - minimum, first quartile ($Q1$), median ($Q2$), third quartile ($Q3$), and maximum.

ATTRIBUTE VALUE DRILL-DOWN FOR AUTO MPG

	cylinders	displacement	origin
Minimum	3	68	1
Q1	4	104.2	1
Median	4	148.5	1
Q3	8	262	2
Maximum	8	455	3

MEASURING DATA VALUE POSITION

- For the attribute ‘displacement’, we can see that
 - the difference between minimum value and Q1 is 36.2 and
 - the difference between Q1 and median is 44.3.
 - the difference between median and Q3 is 113.5 and
 - the difference between Q3 and the maximum value is 193.
- In other words, the larger values are more spread out than the smaller ones.
- This helps in understanding why the value of mean is much higher than that of the median for the attribute ‘displacement’.

MEASURING DATA VALUE POSITION

- Quantiles refer to specific points in a data set which divide the data set into equal parts or equally sized quantities.
- There are specific variants of quantile, the one dividing data set into four parts being termed as quartile.
- Another such popular variant is percentile, which divides the data set into 100 parts.

EXPLORING NUMERICAL DATA

- However, we still cannot ascertain whether there is any outlier present in the data.
- There are two most effective mathematical plots to explore (visualize) numerical data
 - box plot and histogram.

TYPES OF DATA THAT CAN BE MINED

- Flat Files
- Relational Databases
- Data Warehouse
- Transactional Databases
- Multimedia Databases
- Spatial Databases
 - used to describe any data related to or containing information about a specific location on the Earth's surface.

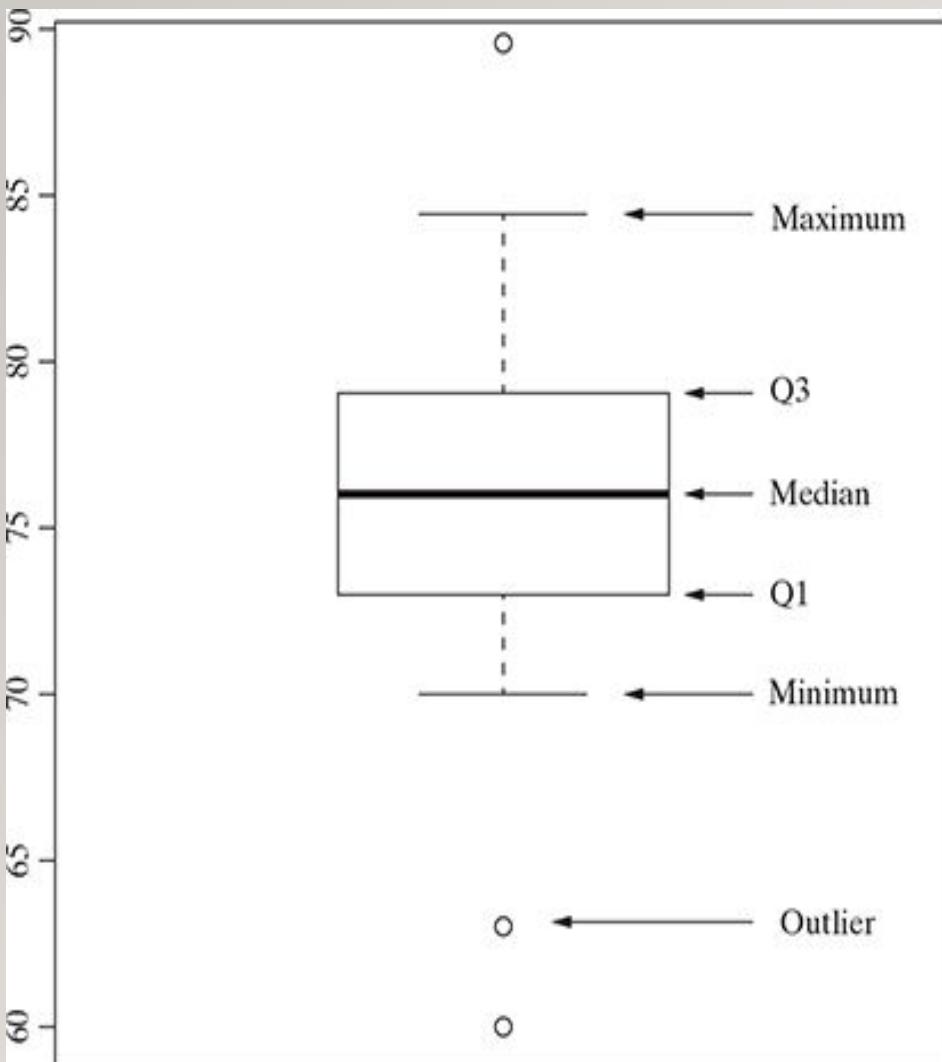
-
- Time Series Databases
 - a sequence of data points collected over an interval of time.
 - Weather data, Rainfall measurements, Temperature readings, Heart rate monitoring (EKG), Brain monitoring (EEG), Quarterly sales, Stock prices
 - World Wide Web(WWW)

EXPLORING (& PLOTTING) NUMERICAL AND CATEGORICAL DATA

BOX PLOTS

- is an extremely effective mechanism to get a one-shot view and understand the nature of the data.
- Also called as whisker plot
- gives a standard visualization of the five-number summary statistics of a data, namely minimum, first quartile (Q1), median (Q2), third quartile (Q3), and maximum

BOX PLOTS



- The central rectangle or the box spans from first to third quartile (i.e. Q1 to Q3), thus giving the inter-quartile range (IQR).
- Median is given by the line or band within the box.**
- The lower whisker extends up to 1.5 times of the inter-quartile range (or IQR) from the bottom of the box, i.e. the first quartile or Q1.
- The upper whisker extends up to 1.5 times of the inter-quartile range (or IQR) from the top of the box, i.e. the third quartile or Q3.
- The data values coming beyond the lower or upper whiskers are the ones which are of unusually low or high values respectively.
- These are the outliers, which may deserve special consideration.

LOWER WHISKER

BOX PLOTS

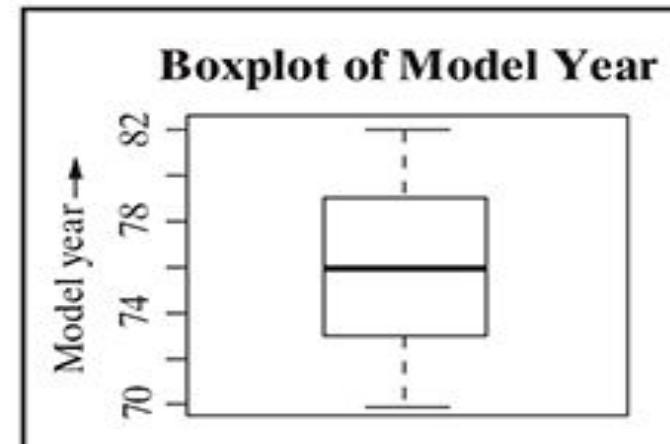
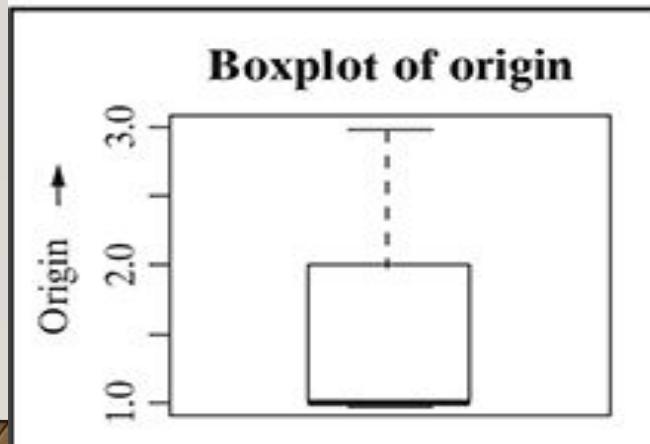
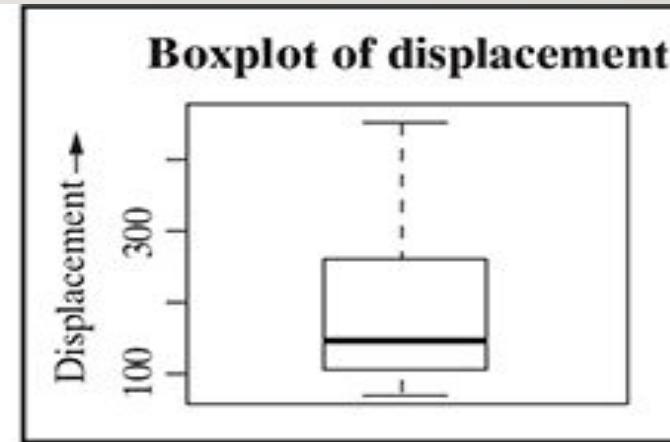
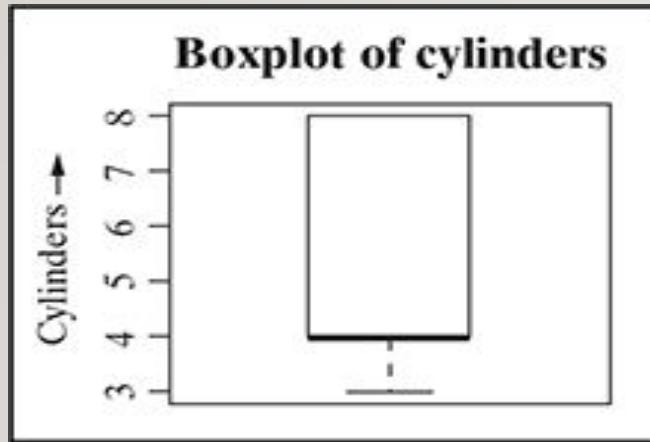
- The lower whisker extends up to 1.5 times of the inter-quartile range (or IQR) from the bottom of the box, i.e. the first quartile or Q1.
- However, the actual length of the lower whisker depends on the lowest data value that falls within ($Q1 - 1.5 \times IQR$).
- Say for a specific set of data, $Q1 = 73$, median = 76 and $Q3 = 79$.
 - Hence, IQR will be 6 (i.e. $Q3 - Q1$).
- So, lower whisker can extend maximum till $(Q1 - 1.5 \times IQR) = 73 - 1.5 \times 6 = 64$.
- However, say there are lower range data values such as 70, 63, and 60.
- So, the lower whisker will come at 70 as this is the lowest data value larger than 64.

UPPER WHISKER

BOX PLOTS

- The upper whisker extends up to 1.5 as times of the inter-quartile range (or IQR) from the top of the box, i.e. the third quartile or Q3.
- Similar to lower whisker, the actual length of the upper whisker will also depend on the highest data value that falls within $(Q3 + 1.5 \times \text{IQR})$.
- For the same set of data mentioned in the above point, upper whisker can extend maximum till $(Q3 + 1.5 \times \text{IQR}) = 79 + 1.5 \times 6 = 88$.
- If there is higher range of data values like 82, 84, and 89.
- So, the upper whisker will come at 84 as this is the highest data value lower than 88.

BOX PLOT OF AUTO MPG ATTRIBUTES

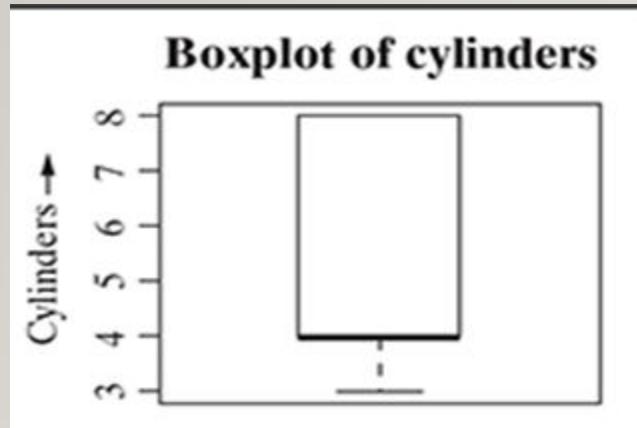


FREQUENCY OF “CYLINDERS” ATTRIBUTE

Cylinders	Frequency	Cumulative Frequency
3	4	4
4	204	$208 (= 4 + 204)$
5	3	$211 (= 208 + 3)$
6	84	$295 (= 211 + 84)$
7	0	$295 (= 295 + 0)$
8	103	$398 (= 295 + 103)$

- Since the total frequency is 398, the first quartile (Q1), median (Q2), and third quartile (Q3) will be at a cumulative frequency 99.5 (i.e. average of 99th and 100th observation), 199 and 298.5 (i.e. average of 298th and 299th observation), respectively. This way Q1 = 4, median = 4 and Q3 = 8.

ANALYSING BOXPLOT OF CYLINDERS



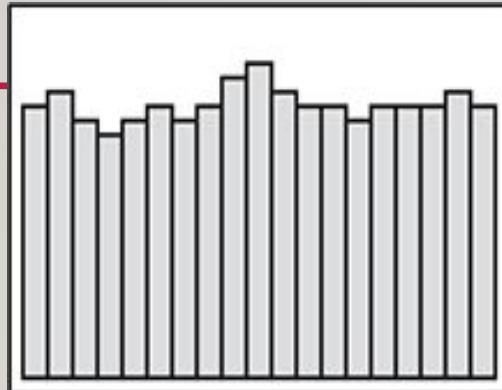
- Since there is no data value beyond 8, there is no upper whisker.
- Also, since both Q1 and median are 4, the band for median falls on the bottom of the box.
- Same way, though the lower whisker could have extended till -2 ($Q1 - 1.5 \times IQR = 4 - 1.5 \times 4 = -2$), in reality, there is no data value lower than 3. Hence, the lower whisker is also short.

HISTOGRAM

- Histogram is another plot which helps in effective visualization of numeric attributes.

- It helps in understanding the distribution of a numeric data into series of intervals, also termed as ‘bins’.
- The histogram is composed of a number of bars, one bar appearing for each of the ‘bins’.
- The height of the bar reflects the total count of data elements whose value falls within the specific bin value, or the frequency.
- Difference between histogram and box plot is
 - The focus of histogram is to plot ranges of data values (acting as ‘bins’), the number of data elements in each range will depend on the data distribution. Based on that, the size of each bar corresponding to the different ranges will vary.
 - The focus of box plot is to divide the data elements in a data set into four equal portions, such that each portion contains an equal number of data elements.

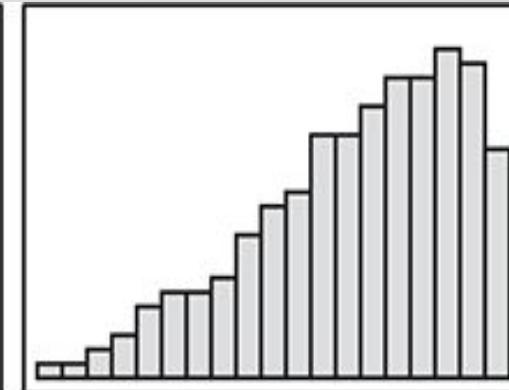
DIFFERENT SHAPES OF HISTOGRAMS DEPENDING ON THE NATURE OF THE DATA, E.G. SKEWNESS



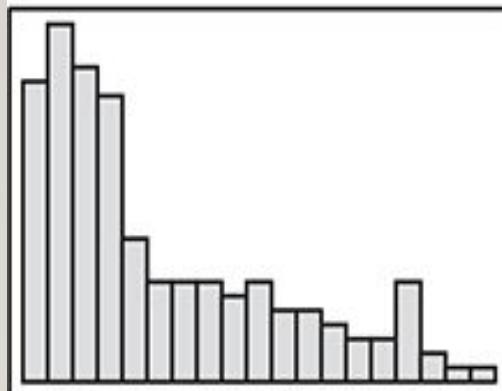
Symmetric, Uniform



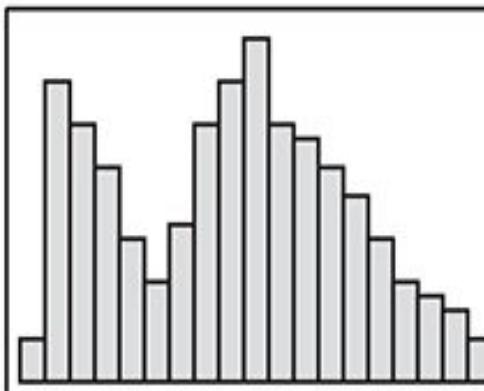
Symmetric, unimodal



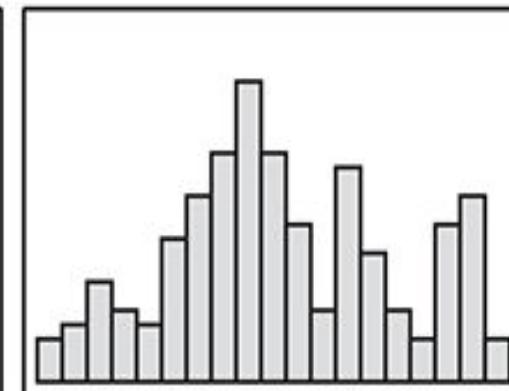
Left skewed



Right skewed

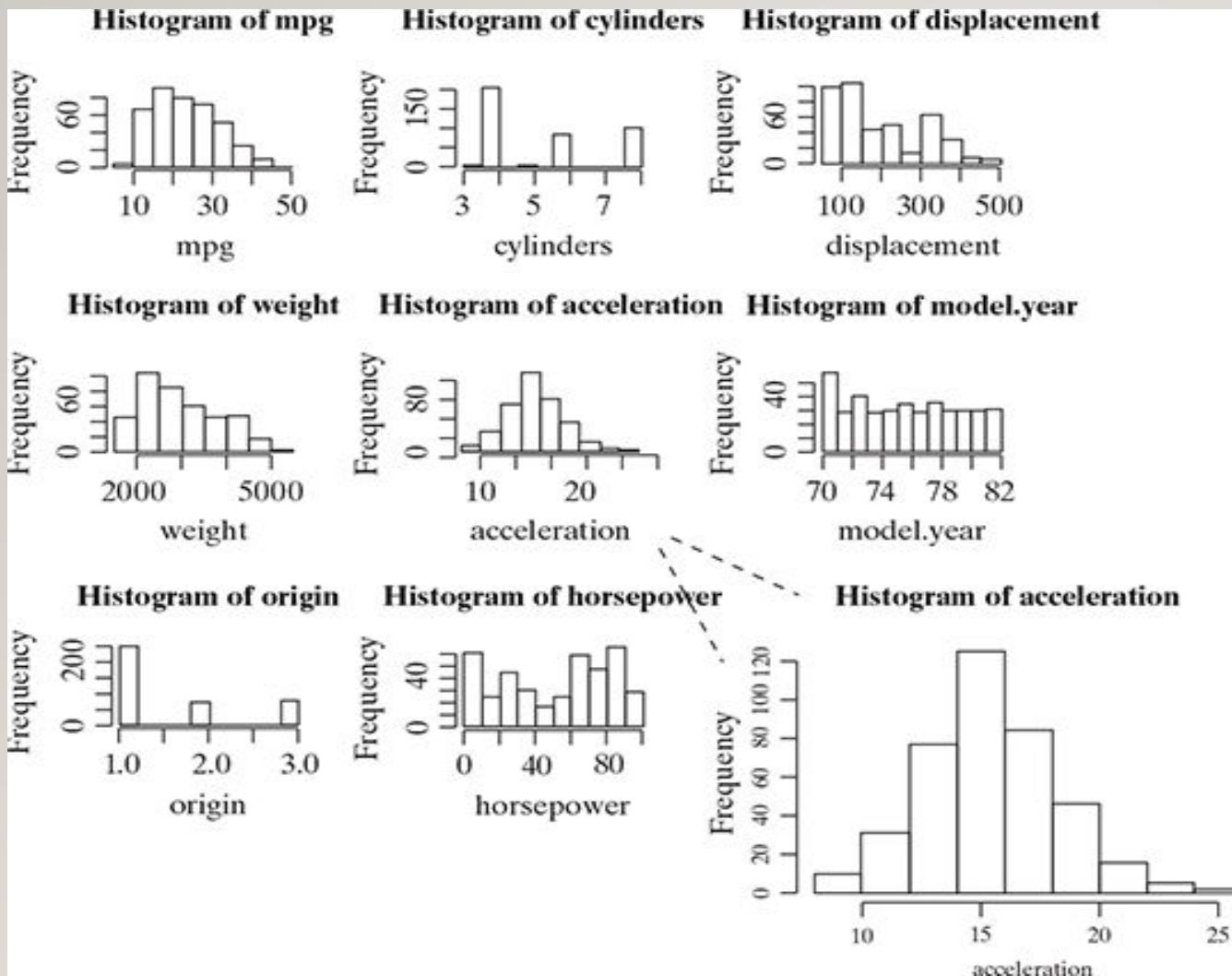


Bimodal



Multimodal

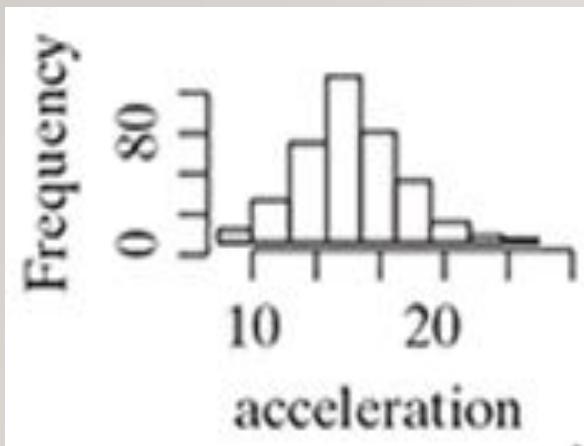
HISTOGRAM AUTO MPG ATTRIBUTES



- The histograms for ‘mpg’ and ‘weight’ are right-skewed.
- The histogram for ‘acceleration’ is symmetric and unimodal,
- The histogram for ‘model.year’ is symmetric and uniform.
- For the remaining attributes, histograms are multimodal in nature.

HISTOGRAM OF ACCELERATION

- Each ‘bin’ represents an acceleration value interval of 2 units.
- So the second bin, e.g., reflects acceleration value of 10 to 12 units.
- The corresponding bar chart height reflects the count of all data elements whose value lies between 10 and 12 units.
- Also, it is evident from the histogram that it spans over the acceleration value of 8 to 26 units.
- The frequency of data elements corresponding to the bins first keep on increasing, till it reaches the bin of range 14 to 16 units. At this range, the bar is tallest in size.
- So we can conclude that a maximum number of data elements fall within this range.
- After this range, the bar size starts decreasing till the end of the whole range at the acceleration value of 26 units.



-
- when the histogram is uniform, as in the case of attribute ‘model. year’, it gives a hint that all values are equally likely to occur.

EXPLORING CATEGORICAL DATA

- how many unique names are there for the attribute ‘car name’ or how many unique values are there for ‘cylinders’ attribute.
- For attribute ‘car name’
 - Chevrolet chevelle malibu
 - Buick skylark 320
 - Plymouth satellite
 - Amc rebel sst
 - Ford torino
 - Ford galaxie 500
 - Chevrolet impala
 - Plymouth fury iii
 - Pontiac catalina
 - Amc ambassador dpl
- For attribute ‘cylinders’
 - 8 4 6 3 5

EXPLORING CATEGORICAL DATA

- to get a table consisting the categories of the attribute and count of the data elements falling into that category.

Attribute	amc	amc ambas-	amc	amc	amc	amc con-	amc	...
Value	ambas-	sador dpl	ambassa-	concord	concord	cord dl 6	gremlin	
Count	1	1	1	1	2	2	4	...

Count of Categories for ‘car name’
Attribute

Attribute	amc	amc ambas-	amc	amc	amc	amc con-	amc	...
Value	ambas-	sador dpl	ambassa-	concord	concord	cord dl 6	gremlin	
Count	1	1	1	1	2	2	4	...

Count of Categories for ‘Cylinders’
Attribute

EXPLORING CATEGORICAL DATA

- to know the proportion (or percentage) of count of data elements belonging to a category.
 - Say, e.g., for the attributes ‘cylinders’, the proportion of data elements belonging to the category 4 is $204 \div 398 = 0.513$, i.e. 51.3%.

Cylinders	Frequency	Cumulative Frequency
3	4	4
4	204	$208 (= 4 + 204)$
5	3	$211 (= 208 + 3)$
6	84	$295 (= 211 + 84)$
7	0	$295 (= 295 + 0)$
8	103	$398 (= 295 + 103)$

EXPLORING CATEGORICAL DATA

Attribute	Amc	Amc	Amc	Amc	Amc	Amc	Amc	...
Value	ambas-	ambassa-	ambassa-	concord	concord	concord	gremlin	
	sador	dor dpl	dor sst		d/l	dl 6		
Count	0.003	0.003	0.003	0.003	0.005	0.005	0.01	...

Proportion of Categories for “car name” Attribute

Attribute	3	4	5	6	8
Value					
Count	0.01	0.513	0.008	0.211	0.259

Proportion of Categories for “Cylinders” Attribute

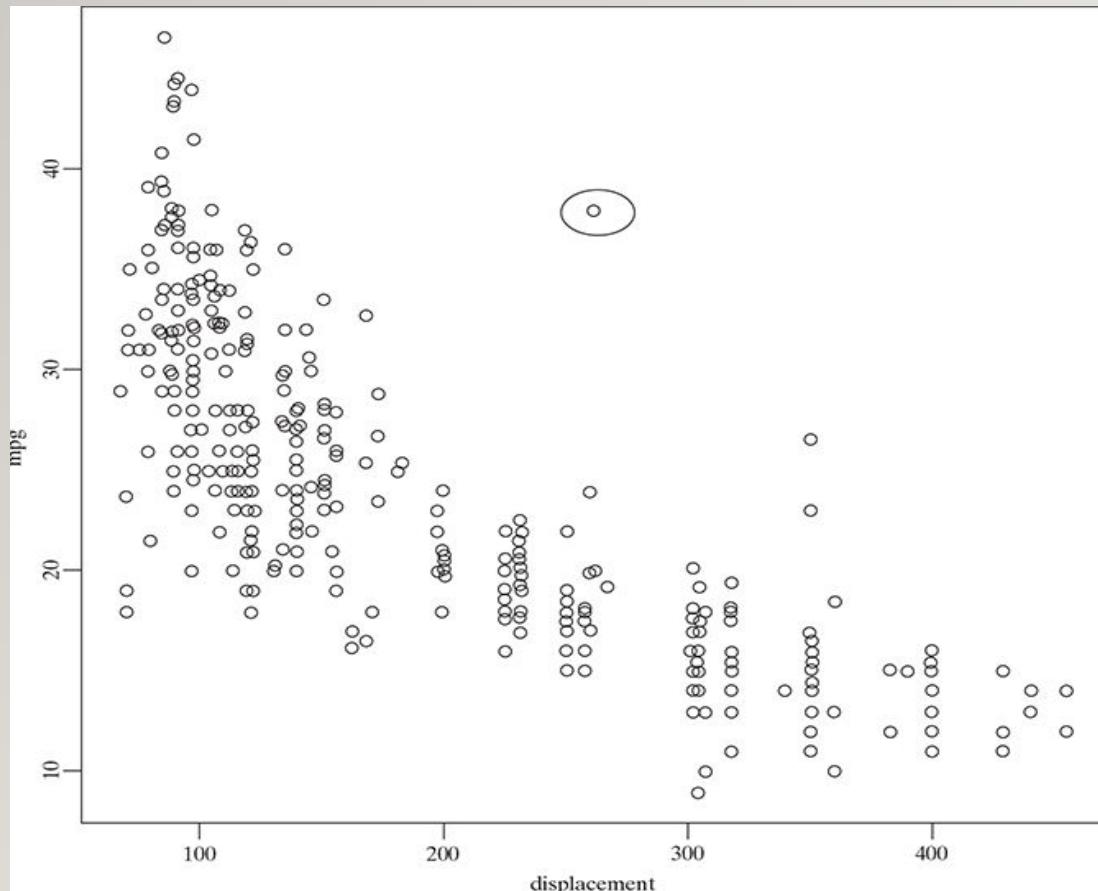
MODE: CATEGORICAL ATTRIBUTES

- “mode” is applicable on categorical attributes.
- Like mean and median, mode is also a statistical measure for central tendency of a data.
- Mode of a data is the data value which appears most often
- For cylinders, since the number of categories is less, we can see that the mode is 4
- For car name, the mode is the category ‘ford pinto’ for which frequency is of highest value 6.

EXPLORING RELATIONSHIP BETWEEN VARIABLES

- A scatter plot helps in visualizing bivariate relationships, i.e. relationship between two variables.
- For example, in a data set there are two attributes – attr_1 and attr_2.
- We want to understand the relationship between two attributes, i.e.
 - with a change in value of one attribute, say attr_1, how does the value of the other attribute, say attr_2, changes.
- We can draw a scatter plot, with attr_1 mapped to x-axis and attr_2 mapped in y-axis.
- So, every point in the plot will have value of attr_1 in the x-coordinate and value of attr_2 in the y-coordinate.
- As in a two-dimensional plot, attr_1 is said to be the independent variable and attr_2 as the dependent variable.

SCATTER PLOT OF 'DISPLACEMENT' AND 'MPG'



- The value of 'mpg' seems to steadily decrease with the increase in the value of 'displacement'
- it can be reviewed by calculating the correlation between the variables.
- An indication of presence of outlier data values.
 - there is one data element which has a mpg of 37 for a displacement of 250.
 - This record is completely different from other data elements having similar displacement value but mpg value in the range of 15 to 25.

TWO-WAY CROSS-TABULATIONS

- Two-way cross-tabulations (also called cross-tab or contingency table) are used to understand the relationship of two categorical attributes in a concise way.
- It has a matrix format that presents a summarized view of the bivariate frequency distribution.
- A cross-tab, very much like a scatter plot, helps to understand how much the data values of one attribute changes with the change in data values of another attribute.

-
- Let's assume the attributes 'cylinders', 'model.year', and 'origin' as categorical and try to examine the variation of one with respect to the other.
 - A 'cylinders' reflects the number of cylinders in a car and assumes values 3, 4, 5, 6, and 8.
 - Attribute 'model.year' captures the model year of each of the car
 - Attribute 'origin' gives the region of the car, the values for origin 1, 2, and 3 corresponding to North America, Europe, and Asia.

CROSS-TABS

Origin \ Model Year	70	71	72	73	74	75	76	77	78	79	80	81	82
1	22	20	18	29	15	20	22	18	22	23	7	13	20
2	5	4	5	7	6	6	8	4	6	4	9	4	2
3	2	4	5	4	6	4	4	6	8	2	13	12	9

Cross-tab for 'Model year' vs.
'Origin'

Cylinders \ Origin	1	2	3
3	0	0	4
4	72	63	69
5	0	3	0
6	74	4	6
8	103	0	0

Cross-tab for 'Cylinders' vs.
'Origin'

- help us understand the number of vehicles per year in each of the regions North America, Europe, and Asia.

- it gives the number of 3, 4, 5, 6, or 8 cylinder cars in every region present in the sample data set.

CROSS-TAB

Cylinders \ Model Year	70	71	72	73	74	75	76	77	78	79	80	81	82
3	0	0	1	1	0	0	0	1	0	0	1	0	0
4	7	13	14	11	15	12	15	14	17	12	25	21	28
5	0	0	0	0	0	0	0	0	1	1	1	0	0
6	4	8	0	8	7	12	10	5	12	6	2	7	3
8	18	7	13	20	5	6	9	8	6	10	0	1	0

Cross-tab for 'Cylinders' vs. 'Model year'

- presents the number of 3, 4, 5, 6, or 8 cylinder cars every year.

DATA QUALITY AND REMEDIATION

- Data quality
 - missing value
 - Outliers
- Reasons
 - Incorrect sample set selection
 - Errors in data collection
- Data remediation
 - by proper sampling technique
 - Handling outliers
 - *Handling missing values*

HANDLING OUTLIERS

- **Remove outliers:** If the number of records which are outliers is not many, a simple approach may be to remove them.
- **Imputation:**
 - Imputation is a method to assign a value to the data elements having outliers.
 - One other way is to impute the value with mean or median or mode.
 - The value of the most similar data element may also be used for imputation.
- **Capping:** For values that lie outside the $1.5 \times |IQR|$ limits, we can cap them by replacing those observations below the lower limit with the value of 5th percentile and those that lie above the upper limit, with the value of 95th percentile.

HANDLING MISSING VALUES

- Eliminate records having a missing value of data elements
- Imputing missing values
 - assign a value to the data elements having missing values.
 - Mean/mode/median is most frequently assigned value.
 - For quantitative attributes
 - all missing values are imputed with the mean, median, or mode of the remaining values under the same attribute.
 - For qualitative attributes
 - all missing values are imputed by the mode of all remaining values of the same attribute.
 - However, another strategy may be identify the similar types of observations whose values are known and use the mean/median/mode of those known values.
- *Estimate missing values*

HANDLING MISSING VALUES

- *Estimate missing values*
 - If there are data points similar to the ones with missing attribute values, then the attribute values from those similar data points can be planted in place of the missing value.
 - For finding similar data points or observations, distance function can be used.
 - For example, let's assume that the weight of a Russian student having age 12 years and height 5 ft. is missing.
 - Then the weight of any other Russian student having age close to 12 years and height close to 5 ft. can be assigned.

POPULAR DATASETS

Iris flower data set	Contains various class of flowers
MNIST database	Images of handwritten digits commonly used to test classification, clustering, and image processing algorithms
ImageNet	Images of 1000 categories
Twitter Sentiment Analysis Dataset	Text data, emoji
BBC News Datasets, Amazon Reviews Dataset	Text data
YouTube Dataset	Video data classification
UCI machine learning repository	

DATA MATRIX

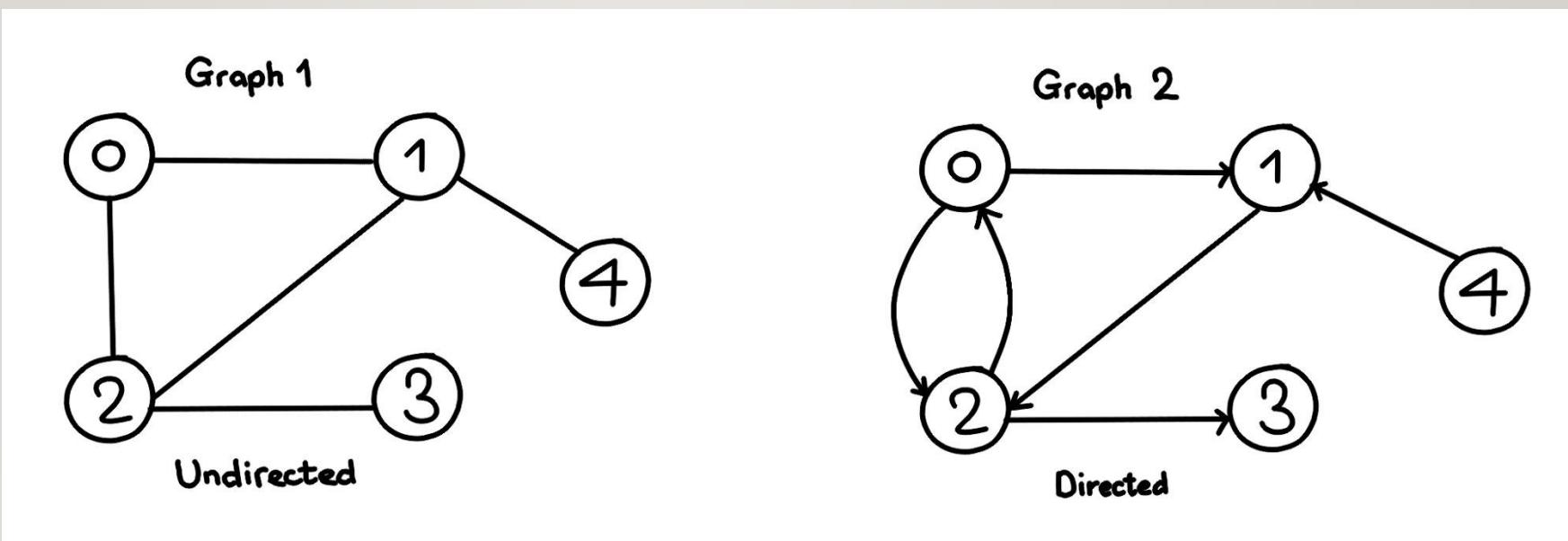
- To be analyzable with statistical tools, data needs to be presented as a ***rectangular data matrix***
- Each column in a data matrix contains a ***variable*** (indicator, measurement, questions in a survey...) and each row an ***observation(response)***

Schematic representation of a *Data Matrix*

Country	Continent	ContNum	GDPperCapita	Variablei
Afghanistan	Asia	1	value	value
Albania	Europe	3	value	value
...
countryn	...	value	value	value

GRAPH DATA

- A graph data structure consists of a finite (and possibly mutable) **set of vertices** (also called nodes or points), together with a set of unordered pairs of these vertices for an undirected graph or a set of ordered pairs for a directed graph



GRAPH BASED DATA MINING

- Graph mining is central to web mining because the web links form a huge graph and mining its properties has a large significance.
- Graph data mining is **used to discover useful information and knowledge from graph data.**

DOCUMENT TERM MATRIX

- A **document-term matrix** is a mathematical matrix that describes the frequency of terms that occur in a collection of documents
- It is also common to encounter the transpose, or **term-document matrix** where documents are the columns and terms are the rows
- They are useful in the field of natural language processing and computational text analysis

DOCUMENT TERM MATRIX - EXAMPLE

	text	analysis	is	fun	I	like	
Text analysis is fun	1	1	1	1	0	0	
I like doing text analysis	1	1	0	0	1	1	
I like puppies, they are fun	0	0	0	1	1	1	
I like this blog post	0	0	0	0	1	1	

SEQUENTIAL DATA

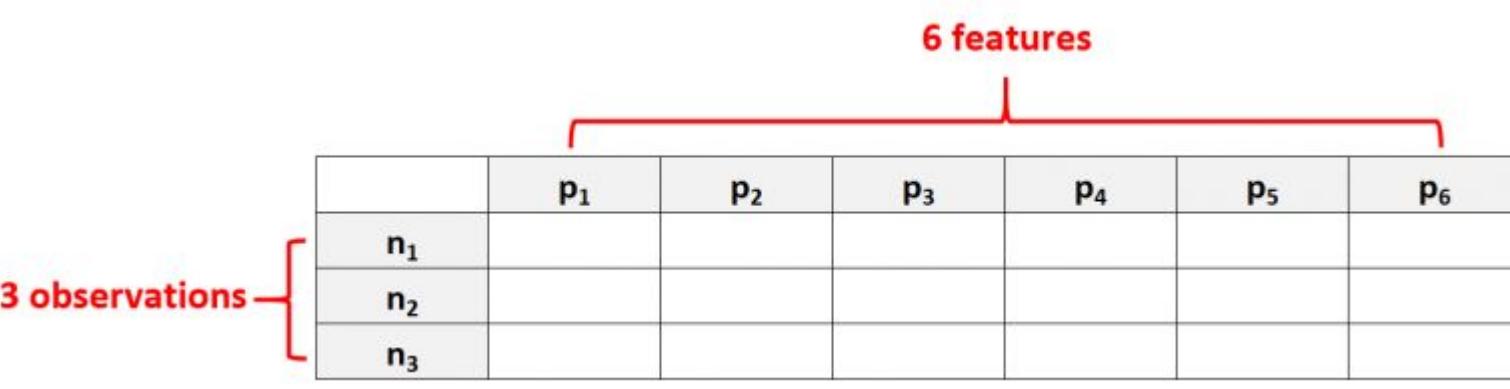
- Whenever the points in the dataset are dependent on the other points in the dataset the data is said to be Sequential data.
- A common example of this is a Timeseries such as a stock price or a sensor data where each point represents an observation at a certain point in time.
- Human action recognition from video data ??? Is this sequential data ?

SPARSE DATA MATRIX

- A sparse matrix is a matrix that is comprised of mostly zero values.
- An example of a smaller sparse matrix might be a word or term occurrence matrix for words in one book against all known words in English.
- This is clearly a waste of memory resources as those zero values do not contain any information.

WHAT IS HIGH DIMENSIONAL DATA?

- **High dimensional data** refers to a dataset in which the number of features p is larger than the number of observations N , often written as $p \gg N$
- For example, a dataset that has $p = 6$ features and only $N = 3$ observations would be considered high dimensional data because the number of features is larger than the number of observations



	p_1	p_2	p_3	p_4	p_5	p_6
n_1						
n_2						
n_3						

WHAT IS NOT HIGH DIMENSIONALITY

- One common mistake people make is assuming that “high dimensional data” simply means a dataset that has a lot of features
- That’s incorrect. A dataset could have 10,000 features, but if it has 100,000 observations then it’s not high dimensional.

WHY IS HIGH DIMENSIONAL DATA A PROBLEM?

- When the number of features in a dataset exceeds the number of observations, we will never have a deterministic answer
- It becomes impossible to find a model that can describe the relationship between the predictor variables and the response variable
- because we don't have enough observations to train the model on

EXAMPLES OF HIGH DIMENSIONAL DATA

EXAMPLE 1: HEALTHCARE DATA

- High dimensional data is common in healthcare datasets where the number of features for a given individual can be massive (i.e. blood pressure, resting heart rate, immune system status, surgery history, height, weight, existing conditions, etc.)
- In these datasets, it's common for the number of features to be larger than the number of observations

The diagram illustrates a high-dimensional healthcare dataset as a table. The columns represent features: Blood pressure, Heart rate, height, weight, and two additional columns indicated by ellipses (...). The rows represent observations for three individuals: Person 1, Person 2, and Person 3. A red bracket labeled "observations" spans the three rows, and another red bracket labeled "features" spans the six columns.

	Blood pressure	Heart rate	height	weight
Person 1						
Person 2						
Person 3						

EXAMPLES OF HIGH DIMENSIONAL DATA

EXAMPLE 2: FINANCIAL DATA

- Number of features for a given stock can be quite large (i.e. PE Ratio, Market Cap, Trading Volume, Dividend Rate, etc.)
- In these datasets, it's common for the number of features to be larger than the number of observations



The diagram illustrates a dataset structure. On the left, the word "observations" is written in red, with a red bracket underneath pointing to three rows in a table. Above the table, the word "features" is written in red, with a red bracket above the table pointing to the first five columns. The table itself has a header row with columns labeled "PE Ratio", "Market Cap", "Volume", "Div. Rate", "...", and "...". Below the header, there are three data rows labeled "Stock #1", "Stock #2", and "Stock #3".

	PE Ratio	Market Cap	Volume	Div. Rate
Stock #1						
Stock #2						
Stock #3						

ISSUES WITH HIGH-DIMENSIONAL DATA (THE CURSE OF DIMENSIONALITY)

- Overfitting (Low biasing and high variance)
- Underfitting (High biasing and low variance)

Bias = Training error

Variance= test error

- Curse of Dimensionality

HOW TO HANDLE HIGH DIMENSIONAL DATA

- There are two common ways to deal with high dimensional data:

1. Choose to include fewer features

The most obvious way to avoid dealing with high dimensional data is to simply include fewer features in the dataset

Drop features with many missing values

Drop features with low variance

Drop features with low correlation with the response variable

HOW TO HANDLE HIGH DIMENSIONAL DATA

2. Use a regularization method

- Principal Components Analysis
- Principal Components Regression
- Ridge Regression
- Lasso Regression

<https://www.youtube.com/watch?v=OFyyWcw2cyM>

<https://github.com/krishnaik06/Dimesnsionality-Reduction/blob/master/01 - Principal%20Component%20Analysis.ipynb>

PRACTICAL SESSION OF WEKA SOFTWARE FOR DATA MINING ON IRIS DATASET



WEKA – IRIS ARFF FILE AND CLASSIFICATION

iris - WordPad

File Edit View Insert Format Help

Min Max Mean SD Class Correlation

sepal length: 4.3 7.9 5.84 0.83 0.7826

sepal width: 2.0 4.4 3.05 0.43 -0.4194

petal length: 1.0 6.9 3.76 1.76 0.9490 (high!)

petal width: 0.1 2.5 1.20 0.76 0.9565 (high!)

9. Class Distribution: 33.3% for each of 3 classes.

@RELATION iris

@ATTRIBUTE sepalwidth REAL

@ATTRIBUTE petallength REAL

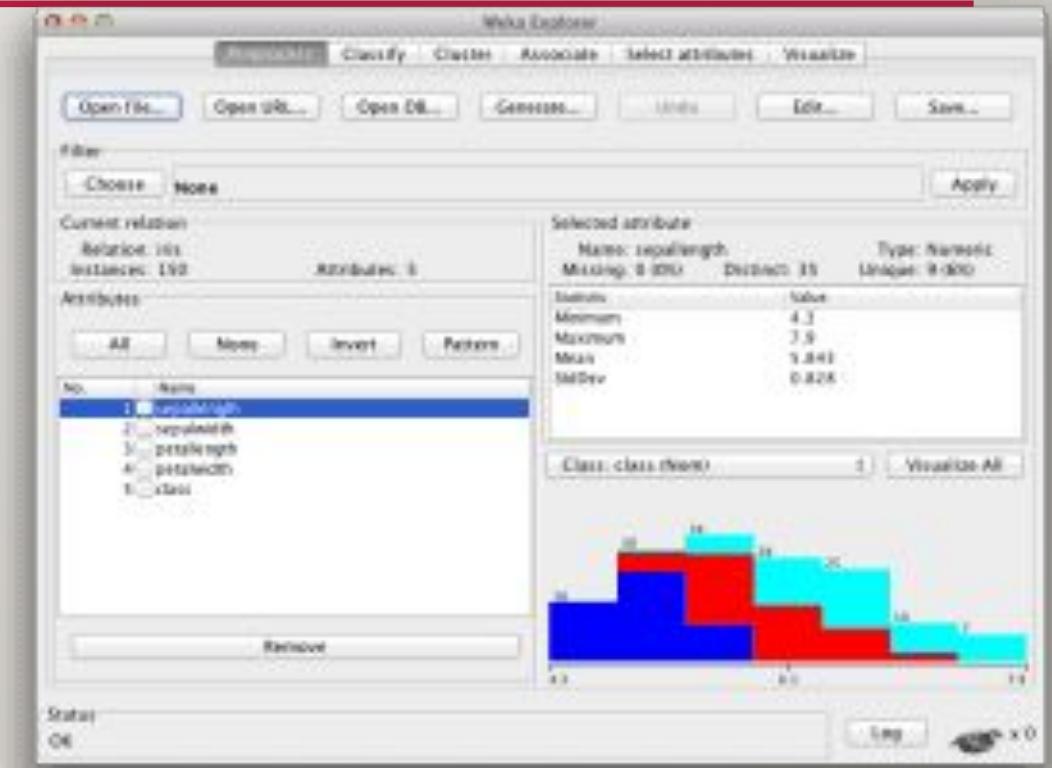
@ATTRIBUTE petalwidth REAL

@ATTRIBUTE class {Iris-setosa, Iris-versicolor, Iris-virginica}

@DATA

5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa

For Help, press F1



Data Preprocessing Tools

Data Preprocessing

- Large datasets
 - Structured Tables, Images, Audio files, Videos, etc..
- Machines don't understand free text, image, or video data as it is, they understand 1s and 0s.
- Step in which the data gets transformed, or *Encoded*, to bring it to such a state that now the machine can easily parse it.
- In other words, the *features* of the data can now be easily interpreted by the algorithm.

Steps of Data Preprocessing

- Not all the steps are applicable for each problem
 - Data Quality Assessment
 - Feature Aggregation
 - Feature Discretization
 - Feature Sampling
 - Dimensionality Reduction
 - Feature Encoding
 - Feature Scaling

Data Quality Assessment

- Missing values
- Outliers
- Inconsistent values
- Duplicate values

Feature Aggregation

- To take the aggregated values in order to put the data in a better perspective.
- Example
 - day-to-day transactions of a product to record the daily sales of that product in various store locations over the year.
 - Aggregating the transactions to single store-wide monthly or yearly transactions will help us reducing hundreds or potentially thousands of transactions that occur daily at a specific store, thereby reducing the number of data objects.
- Why?
 - This results in reduction of memory consumption and processing time
 - Aggregations provide us with a high-level view of the data as the behaviour of groups or aggregates is more stable than individual data objects

Feature Discretization

- transforms the data into sets of small intervals
- For example, putting people in categories “young”, “middle age”, “senior” rather than working with continuous age values.
- Discretization helps to improve efficiency.

Feature Sampling

- Sampling is a very common method for selecting a subset of the dataset that we are analyzing.
- In most cases, working with the complete dataset can turn out to be too expensive considering the memory and time constraints.
- The key principle here is
 - the sampling should be done in such a manner that the sample generated should have approximately the same properties as the original dataset, meaning that the sample is *representative*.
- *Simple Random Sampling*
 - Sampling without Replacement
 - Sampling with Replacement

Feature Sampling

- Simple Random Sampling
 - imbalanced dataset
 - number of instances of a class(es) are significantly higher than another class(es),
- Stratified Sampling
 - which begins with predefined groups of objects

Dimensionality Reduction

- Most real world datasets have a large number of features.
- For example, consider an image processing problem, we might have to deal with thousands of features, also called as *dimensions*.
- Dimensionality reduction aims to reduce the number of features
 - but not simply by selecting a sample of features from the *feature-set*: Feature Subset Selection or simply Feature Selection.
- Conceptually, *dimension* refers to the number of geometric planes the dataset lies in
 - More the number of such planes, more is the complexity of the dataset.

Dimensionality Reduction

- Dimension reduction maps the dataset to a lower-dimensional space
- The basic objective of techniques is to reduce the dimensionality of a dataset by creating new features which are a combination of the old features.
- Two widely accepted techniques
 - Principal Component Analysis
 - Singular Value Decomposition

Feature Encoding

- Performing transformations on the data such that it can be easily accepted as input for machine learning algorithms while still retaining its original meaning.
- There are some general norms or rules which are followed when performing feature encoding.

General norms or rules which are followed when performing feature encoding

- Nominal
 - Any one-to-one mapping can be done which retains the meaning.
 - For instance, a permutation of values like in One-Hot Encoding.
- Ordinal
 - An order-preserving change of values.
 - Unhappy, Happy, Very Happy- 1,2,3
- Interval
 - $\text{new_value} = a * \text{old_value} + b$, a and b being constants.
 - For example, Fahrenheit and Celsius scales, which differ in their Zero values size of a unit
- Ratio
 - These variables can be scaled to any particular measures, while maintaining the meaning and ratio of their values.
 - $\text{new_value} = a * \text{old_value}$
 - length can be measured in meters or feet, money can be taken in different currencies.

One Hot Encoding

Dummy vars trap

One Hot Encoding

Dummy Vars

productivity	Years of Experience	No of years of education	Last Degree
Pr1	Ex1	Ed1	M.Sc
Pr2	Ex2	Ed2	M.Tech
Pr3	Ex3	Ed3	M.Tech

$$Y = b_0 + b_1 * x_1 + b_2 * x_2 + b_3 * D_1 \text{ (for M.Sc.)}$$

Dummy Vars

productivity	Years of Experience	No of years of education	Last Degree	M.Sc.	M.Tech
Pr1	Ex1	Ed1	M.Sc	1	0
Pr2	Ex2	Ed2	M.Tech	0	1
Pr3	Ex3	Ed3	M.Tech	0	1

$$Y = b_0 + b_1 * x_1 + b_2 * x_2 + b_3 * D_1 \text{ (for M.Sc.)}$$

Dummy Var Trap

- If we will take var for each dummy vars like
 - $Y = b_0 + b_1 * x_1 + b_2 * x_2 + b_3 * D_1$ (for M.Sc.) + $b_4 * D_2$ (for M.Tech.)
 - We will duplicate the vars because $D_2 = D_1 - 1$
 - Model will not distinguish the effect of D1 from the effect of D2.
 - This is called the dummy var trap
 - We can not have constant and all dummy vars at the same time.
 - Whenever you build the model, always omit one dummy var.

Feature scaling

- Some machine learning algorithms are sensitive to feature scaling while others are virtually invariant to it.
- We need to perform Feature Scaling when we are dealing with
 - Gradient Descent Based algorithms and
 - Linear and Logistic Regression, Neural Network
 - Distance-based algorithms
 - KNN, K-means, SVM
 - As these algorithms are very sensitive to the range of the data points.

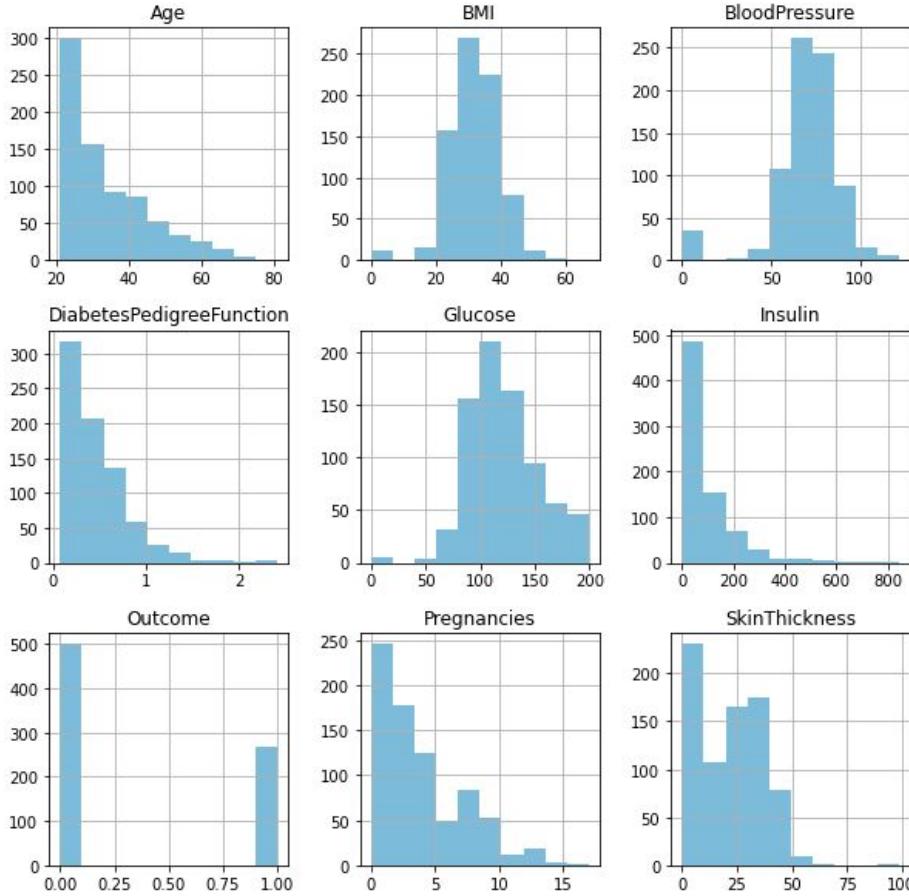
Feature Scaling

Standardization

- $X' = \frac{x - \mu}{\sigma}$
- Values are shifted and rescaled so that they end up ranging between -3 and +3.
- the values are centered around the mean with a unit standard deviation.
- converts the data to mean of 0 and standard deviation of 1.
- Works all the time

Normalization

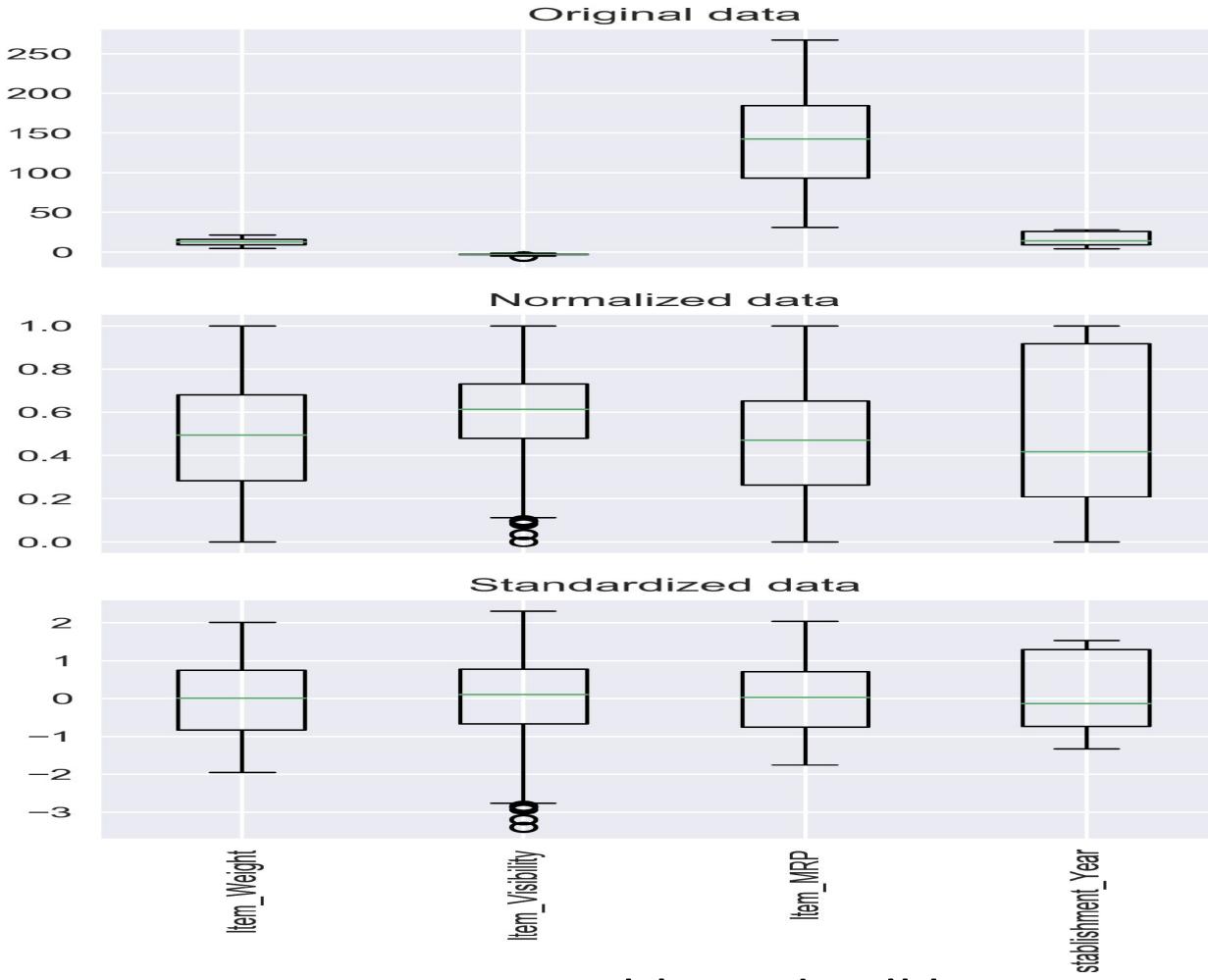
- $X' = \frac{x - X_{min}}{X_{max} - X_{min}}$
- values are shifted and rescaled so that they end up ranging between 0 and 1.
- It is also known as Min-Max scaling.
- Is recommended when we have normal distribution in most of the features.



Gaussian Distribution — BMI, BloodPressure, Glucose.

Non-Gaussian Distribution — Age,
DiabetesPedigreeFunction, Insulin, Pregnancies,
SkinThickness

Normalize Non-Gaussian features and Standardize Gaussian-like features



features are now more comparable and will have a similar effect on the learning models

Feature Scaling Requirement

- Gradient Descent Based Algorithms
 - linear regression, logistic regression, neural network, etc. that use gradient descent as an optimization technique require data to be scaled
 - Having features on a similar scale can help the gradient descent converge more quickly towards the minima
- Distance-Based Algorithms
 - KNN, K-means and SVM
 - are most affected by the range of features.
 - they are using distances between data points to determine their similarity.
 - there is a chance that higher weightage is given to features with higher magnitude.
- Tree-Based Algorithms
 - are fairly insensitive to the scale of the features.

Framework

- Importing the libraries
- Importing the dataset
- Taking care of missing data
- Encoding categorical data
 - Encoding the Independent Variable
 - Encoding the Dependent Variable
- Splitting the dataset into the Training set and Test set
- Feature Scaling

Importing the libraries

- import numpy as np
- import matplotlib.pyplot as plt
- import pandas as pd

Importing the dataset

- `dataset = pd.read_csv('Data.csv')`
- `X = dataset.iloc[:, :-1].values`
- `y = dataset.iloc[:, -1].values`
- `print(X)`
 - `[['France' 44.0 72000.0] ['Spain' 27.0 48000.0] ['Germany' 30.0 54000.0] ['Spain' 38.0 61000.0] ['Germany' 40.0 nan] ['France' 35.0 58000.0] ['Spain' nan 52000.0] ['France' 48.0 79000.0] ['Germany' 50.0 83000.0] ['France' 37.0 67000.0]]`
- `print(y)`
 - `['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']`

Dataset (data.csv)

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40		Yes
France	35	58000	Yes
Spain		52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

Taking care of missing data

- from sklearn.impute import SimpleImputer
- imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
- imputer.fit(X[:, 1:3])
- X[:, 1:3] = imputer.transform(X[:, 1:3])
- print(X)
 - [['France' 44.0 72000.0] ['Spain' 27.0 48000.0] ['Germany' 30.0 54000.0] ['Spain' 38.0 61000.0] ['Germany' 40.0 63777.77777777778] ['France' 35.0 58000.0] ['Spain' 38.77777777777778 52000.0] ['France' 48.0 79000.0] ['Germany' 50.0 83000.0] ['France' 37.0 67000.0]]

Encoding categorical data:

Encoding the Independent Variable

- from sklearn.compose import ColumnTransformer
- from sklearn.preprocessing import OneHotEncoder
- ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
- X = np.array(ct.fit_transform(X))

- print(X)
 - [[1.0 0.0 0.0 44.0 72000.0] [0.0 0.0 1.0 27.0 48000.0] [0.0 1.0 0.0 30.0 54000.0] [0.0 0.0 1.0 38.0 61000.0] [0.0 1.0 0.0 40.0 63777.7777777778] [1.0 0.0 0.0 35.0 58000.0] [0.0 0.0 1.0 38.77777777777778 52000.0] [1.0 0.0 0.0 48.0 79000.0] [0.0 1.0 0.0 50.0 83000.0] [1.0 0.0 0.0 37.0 67000.0]]

Encoding categorical data: Encoding the Dependent Variable

- from sklearn.preprocessing import LabelEncoder
- le = LabelEncoder()
- y = le.fit_transform(y)
- print(y)
 - [0 1 0 0 1 1 0 1 0 1]

Splitting the dataset into the Training set and Test set

- `from sklearn.model_selection import train_test_split`
- `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)`
- `print(X_train)`
 - `[[0.0 0.0 1.0 38.77777777777778 52000.0] [0.0 1.0 0.0 40.0 63777.7777777778] [1.0 0.0 0.0 44.0 72000.0] [0.0 0.0 1.0 38.0 61000.0] [0.0 1.0 27.0 48000.0] [1.0 0.0 0.0 48.0 79000.0] [0.0 1.0 0.0 50.0 83000.0] [1.0 0.0 0.0 35.0 58000.0]]`
- `print(X_test)`
 - `[[0.0 1.0 0.0 30.0 54000.0] [1.0 0.0 0.0 37.0 67000.0]]`

Splitting the dataset into the Training set and Test set

- `print(y_train)`
 - [0 1 0 0 1 1 0 1]
- `print(y_test)`
 - [0 1]

Feature Scaling

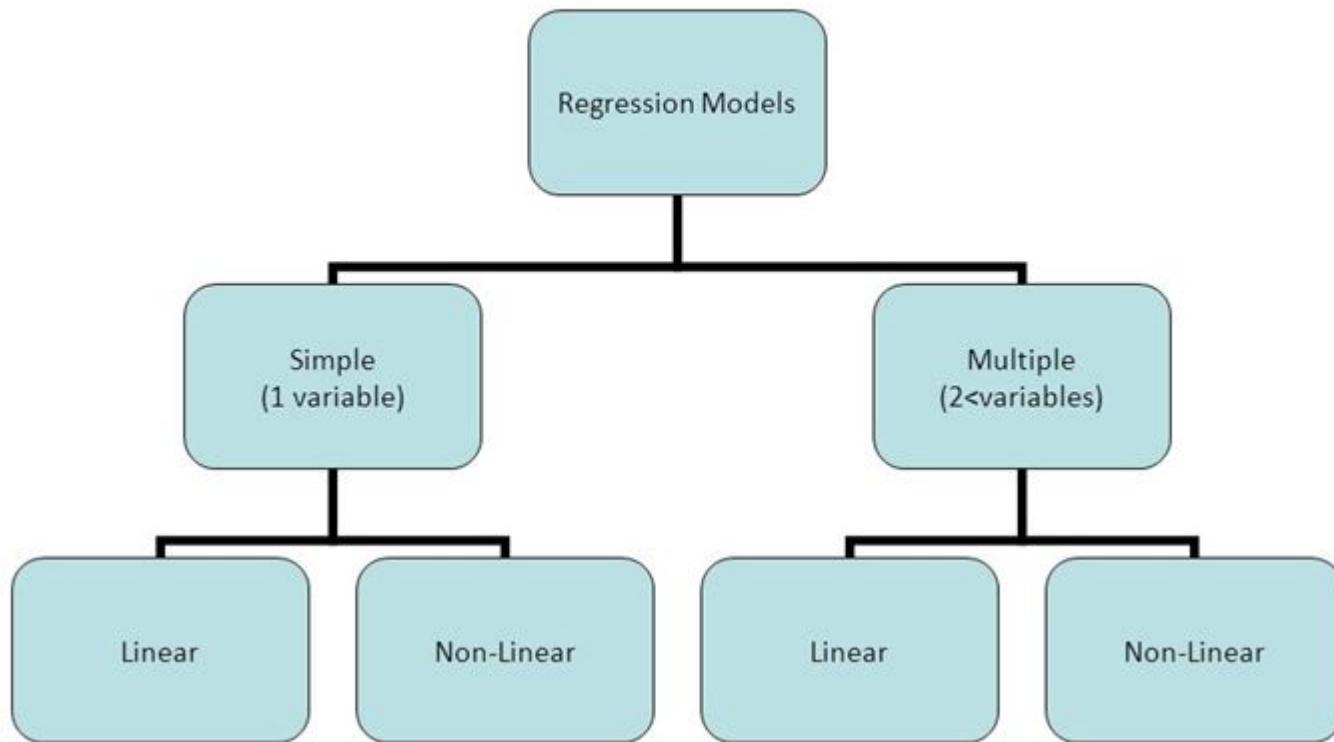
- from sklearn.preprocessing import StandardScaler
- sc = StandardScaler()
- X_train[:, 3:] = sc.fit_transform(X_train[:, 3:])
- X_test[:, 3:] = sc.transform(X_test[:, 3:])

Note: for test set we will use same scaling so we will call only transform method in place of fit_transform

- print(X_train)
 - [[0.0 0.0 1.0 -0.19159184384578545 -1.0781259408412425] [0.0 1.0 0.0 -0.014117293757057777 -0.07013167641635372] [1.0 0.0 0.0 0.566708506533324 0.633562432710455] [0.0 0.0 1.0 -0.30453019390224867 -0.30786617274297867] [0.0 0.0 1.0 -1.9018011447007988 -1.420463615551582] [1.0 0.0 0.0 1.1475343068237058 1.232653363453549] [0.0 1.0 0.0 1.4379472069688968 1.5749910381638885] [1.0 0.0 0.0 -0.7401495441200351 -0.5646194287757332]]
- print(X_test)
 - [[0.0 1.0 0.0 -1.4661817944830124 -0.9069571034860727] [1.0 0.0 0.0 -0.44973664397484414 0.2056403393225306]]

Simple Linear Regression

Types of Regression Model



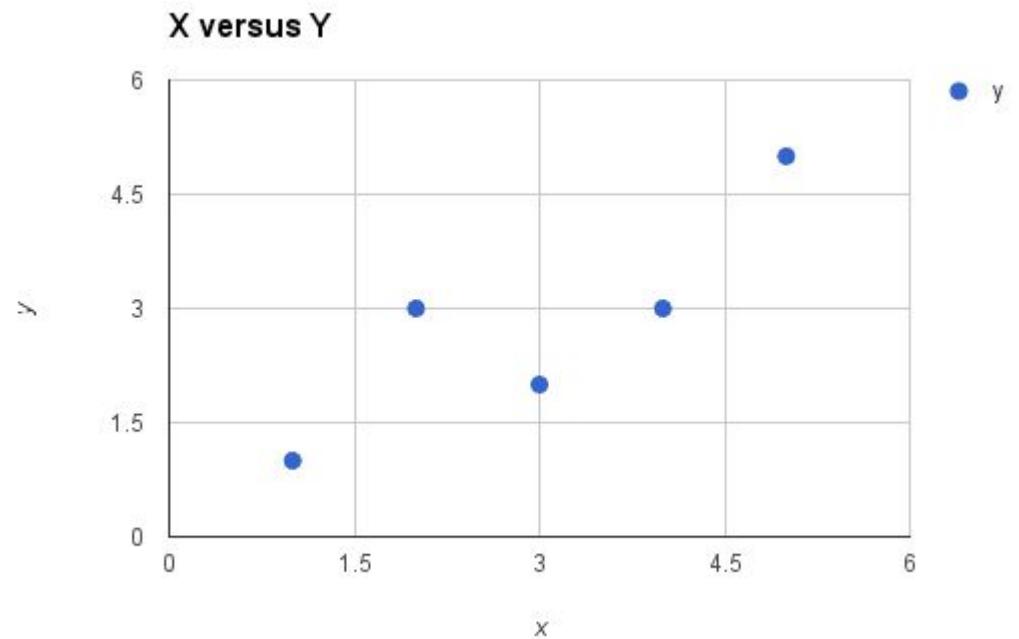
- Regression models are built for 2 reasons:
 1. either to explain the relationship between an exposure and an outcome
 - We will refer to these as *explanatory models*
 2. or to predict the outcome based on a set of independent variables
 - We will refer to these as *prediction models*

Simple Linear Regression

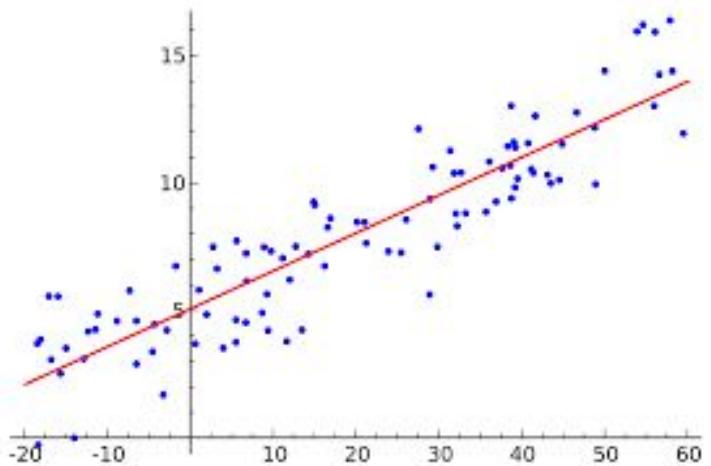
- is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables
- One variable, denoted x , is regarded as the **predictor, explanatory, or independent** variable.
- The other variable, denoted y , is regarded as the **response, outcome, or dependent** variable.
- Not used to study deterministic relationship
 - Circumference = $\pi \times$ diameter
 - Fahr = $9/5$ Cels + 32
- Used to study **statistical relationships**, in which the relationship between the variables is not perfect.

Simple linear regression

- we want to model our data as follows:
 - $y = b_0 + b_1 * x$
- y- dependent var
- x- independent var
- b_1 - coefficient (feature weights)
 - need to estimate
 - Population slope
- b_0 - constant-
 - need to estimate
 - b_0 is called the intercept
- b_0 & b_1 are model's parameters



Constant & Coefficient meaning



- Salary- dependent var
- No of experience- independent var
- Constant- if no of experience is 0, it tells about salary , sometimes may not be valid for the problem i.e. $x=0$ may not be relevant for the problem like if x is height and y is weight.
- Coefficient (slope)- as the experience grows how fast salary will increase

- The goal is to find the best estimates for the coefficients to minimize the errors in predicting y from x .
- There may not exist the complete fit to a straight line so we can assume that there is an error (noise) in the data
 - $y = b_0 + b_1 * x + \epsilon$ \square random error
- We assume that this is the function from which the points are generated so we can think of there is an underlying line from which these points are generated after accounting for some error.
- We are assuming Expected value of $E(Y|X)$ follows this equation. This equation is the equation of the population line i.e. from which examples are actually drawn. So $E(Y|X) = b_0 + b_1 * x$
- And we assume this error has mean of 0 and some standard deviation.
- and given the data points we are trying to find out b_0 and b_1 that is b_0 Hat and b_1 hat so $y = b_0$ hat + b_1 hat * x is the equation that we are trying to come up with that is the estimate for actual target function ($y = b_0 + b_1 * x + \epsilon$).
- We will optimize certain things to get this estimate w.r.t the training examples.
 - We try to get the values of b_0 hat and b_1 hat so that the sum of squared error is minimized and then the equation $y = b_0$ hat + b_1 hat * x is called least squares line.
- So we will see how for the given points we will come up with the least square line.

Assumptions about the error

- The given data points may not form a perfect line so
- w.r.t. every example there is the value of the error
- We will make some assumptions about the error
 - ϵ_i is a normally distributed random var, with mean zero and variance σ^2 so this type of noise is Gaussian noise.
 - $E(\epsilon_i) = 0$.
 - $\sigma(\epsilon_i) = \sigma$
 - Errors ($\epsilon_1, \epsilon_2, \epsilon_3, \dots$) are uncorrelated and independent.

$$d1 \ y1 = b_0 + b_1 * x1 + \epsilon_1$$

$$d2 \ y2 = b_0 + b_1 * x2 + \epsilon_2$$

$$d3 \ y3 = b_0 + b_1 * x3 + \epsilon_3$$

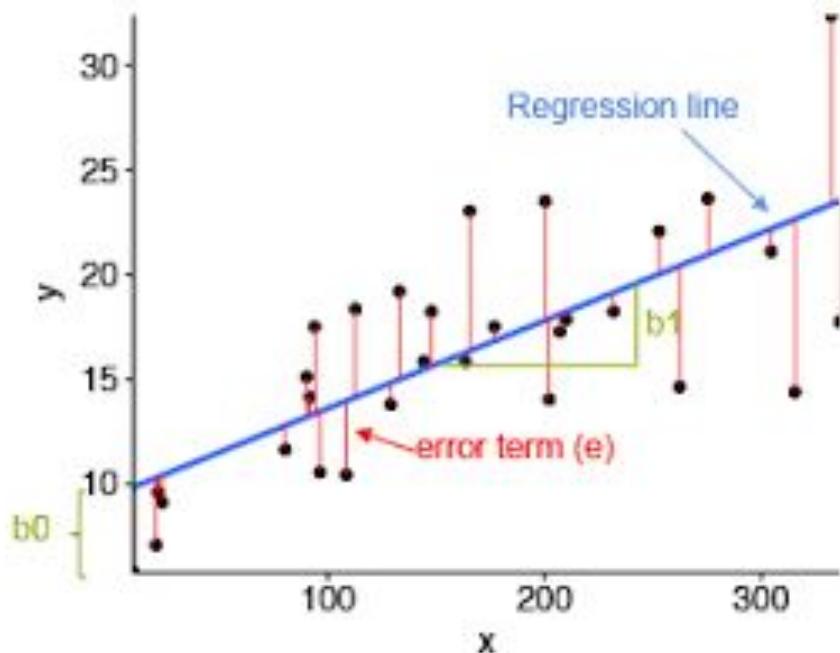
.

.

Consequence of assumptions on error on y

- $y_i = b_0 + b_1 * x_i + \epsilon_i$
- $E(y_i) = E(b_0 + b_1 * x_i + \epsilon_i) = b_0 + b_1 * x_i$
- $V(y_i) = V(b_0 + b_1 * x_i + \epsilon_i) = V(\epsilon_i) = \sigma^2$
- ϵ_i is a normally distributed random var, with mean zero and variance σ^2 and all ϵ_i are independent.
- y_i is a normally distributed random var, with mean $b_0 + b_1 * x_i$ and variance σ^2 and all y_i are independent.
- Given a dataset, while fitting a simple linear regression model, how to check if the dataset is satisfying basic assumptions- **model adequacy checking**

Least squares estimation of parameters



- prediction error/residual error
 - $y_i - y'_i$
- Finds a line which has smallest sum of square least square regression line
 - Sum $(y_i - y'_i)^2$
- Goal
 - minimize the sum of the squared prediction errors
- Why square?
 - if we didn't square the prediction error to get the positive and negative prediction errors would cancel each other out when summed, always yielding 0.
- Line fitted by least square is the one that makes the sum of squares of all vertical discrepancies as small as possible.

Infinite possible lines- Solution?

- We'd have to implement the above procedure for an infinite number of possible lines
 - clearly, an impossible task
- How to choose that line → using algorithm
- We minimize the equation for the sum of the squared prediction errors:
 - $Q = \sum_{i=1}^n (y_i - (\hat{b}_0 + \hat{b}_1 x_i))^2$
 - So how to learn the estimated values of parameters \hat{b}_0 and \hat{b}_1 i.e. \hat{b}_1 and \hat{b}_2
 - Many solutions
 - Normal Equation Method
 - Or we can come up with the iterative solution- Gradient Descent Method

Normal Method

- The least square estimator of b_0 and b_1 i.e. b_0 hat and b_1 hat must satisfy normal equations: $S = \text{Sum}(y_i - b_0 \text{ hat} - b_1 \text{ hat } x_i)^2$
 - $\partial S / \partial b_0 = -2 \sum (y_i - b_0 \text{ hat} - b_1 \text{ hat } x_i) = 0$
 - $\partial S / \partial b_1 = -2 \sum x_i (y_i - b_0 \text{ hat} - b_1 \text{ hat } x_i) = 0$
- So the estimator b_0 hat and b_1 hat are solutions of the equations

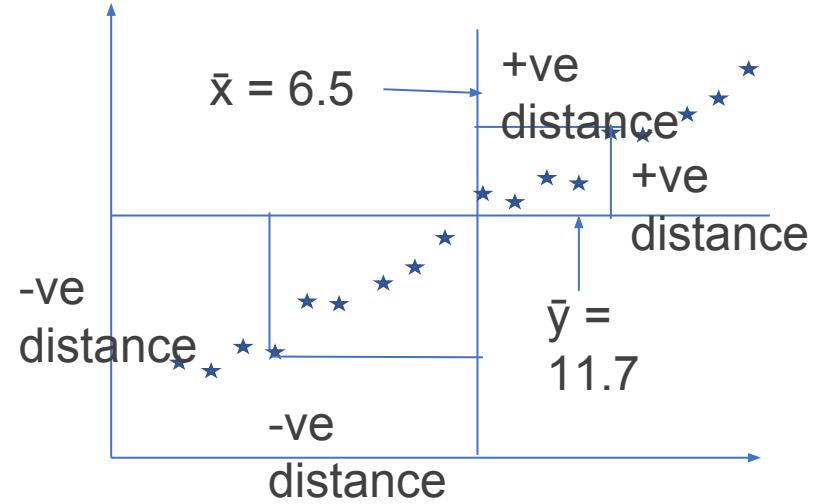
Normal Method

- For the 2-d problem
 - $y = b_0 + b_1 * x$
 - To get the "**least squares estimates**" for b_0 and b_1 which minimize the objective function we take the partial derivative of objective function (SSE) with respect to b_0 and b_1 , set to 0, and solve for b_0 and b_1
 - $b_0 = \bar{y} - b_1 \bar{x}$
and
 - $b_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$

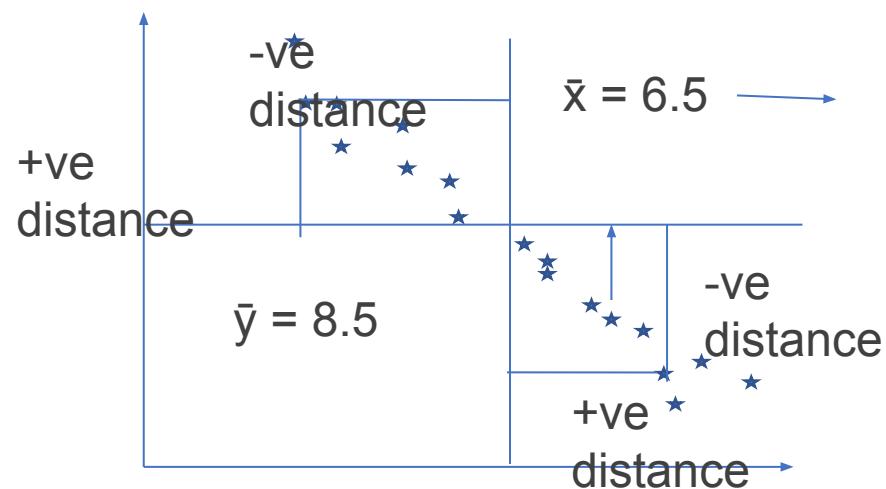
Slope

- Denominator of slope equation is necessarily positive since
 - it only involves summing positive terms.
- Therefore, the sign of the slope b_1 is solely determined by the numerator.
- The numerator tells us, for each data point, to sum up the product of two distances
 - the distance of the x -value from the mean of all of the x -values and
 - the distance of the y -value from the mean of all of the y -values.

Numerator- Positive and Negative cases



So multiplication of $x_i - \bar{x}$ and $y_i - \bar{y}$ will always be positive.



So multiplication of $x_i - \bar{x}$ and $y_i - \bar{y}$ will always be negative.

Assumptions of a linear regression

- Linearity
 - There must be a linear relationship between the outcome variable and the independent variables.
- Homoscedasticity
 - refers to a condition in which the variance of the residual, or error term, in a regression model is constant.
 - That is, the error term does not vary much as the value of the predictor variable changes.
- Multivariate normality
 - Multiple regression assumes that the residuals are normally distributed.
- Independence of errors
- Lack of (No) multicollinearity
 - Multiple regression assumes that the independent variables are not highly correlated with each other. This assumption is tested using Variance Inflation Factor (VIF) values.

Pros and Cons

- Works on any size of dataset, gives information about relevance of features
- The Linear Regression Assumptions need to be checked

Linear Regression

$$\min \sum_{i=1}^m (y_i - \hat{y}_i)^2 = \sum_{i=1}^m (y_i - (\hat{\mathbf{w}} \cdot \mathbf{x}_i + \hat{b}))^2$$



- To avoid over-fitting, a regularization term can be introduced (minimize a magnitude of w)

- Lasso

$$\min \sum_{i=1}^m (y_i - \mathbf{w} \cdot \mathbf{x}_i - b)^2 + C \sum_{j=1}^n |w_j|$$

- Ridge Regression

$$\min \sum_{i=1}^m (y_i - \mathbf{w} \cdot \mathbf{x}_i - b)^2 + C \sum_{j=1}^n |\mathbf{w}_j|^2$$

Years of Experience	Salary in 1000\$
2	15
3	28
5	42
13	64
8	50
16	90
11	58
1	8
9	54

Years of Experience	Salary (in 1000\$)	Error $y_i - \bar{y}$	Error ²
2	15	$15 - 45.44 = -30.44$	926.59
3	28	$28 - 45.44 = -17.44$	304.15
5	42	$42 - 45.44 = -3.44$	11.83
13	64	$64 - 45.44 = 18.56$	344.47
8	50	$50 - 45.44 = 4.56$	20.79
16	90	$90 - 45.44 = 44.56$	1985.59
11	58	$58 - 45.44 = 12.56$	157.75
1	8	$8 - 45.44 = -37.44$	1401.75
9	54	$54 - 45.44 = 8.56$	73.27
$\bar{y} = 45.44$		SSE = 5226.19	

The sum of squared errors SSE output is 5226.19. To do the best fit of line intercept, we need to apply a linear regression model to reduce the SSE value at minimum as possible.

- To identify a slope intercept, we use the equation
- $y = mx + b$,
- ‘m’ is the slope
- ‘x’ → independent variables
- ‘b’ is intercept
- We will use Ordinary Least Squares method to find the best line
intercept (b) slope (m)

$$m = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

$$b = \bar{y} - m * \bar{x}$$

- We need to calculate slope ‘m’ and line intercept ‘b’. Below is the simpler table to calculate those values.

Years of Experience x_i	Salary (in 1000\$) y_i	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$	$(x_i - \bar{x})^2$
2	15	-5.56	-30.44	169.24	30.91
3	28	-4.56	-17.44	79.53	20.79
5	42	-2.56	-3.44	8.81	6.55
13	64	5.44	18.56	100.97	29.59
8	50	0.44	4.56	2.01	0.19
16	90	8.44	44.56	376.09	71.23
11	58	3.44	12.56	43.21	11.83
1	8	-6.56	-37.44	245.61	43.03
9	54	1.44	8.56	12.33	2.07
$\bar{x} = 7.56$	$\bar{y} = 45.44$			$\sum = 1037.8$	$\sum = 216.19$

$$m = 1037.8 / 216.19$$

$$m = 4.80$$

$$b = 45.44 - 4.80 * 7.56 = 9.15$$

$$\text{Hence, } y = mx + b \rightarrow 4.80x + 9.15$$

$$y = 4.80x + 9.15$$

- Let us calculate SSE again by using our output equation.
- Now Sum of Squared Error got reduced significantly from 5226.19 to 245.38.

Years of Experience x	Salary (in 1000\$) y	$\bar{y} = mx + b$ $\bar{y} = 4.79x + 9.18$	Error $y_i - \bar{y}$	Error ²
2	15	18.76	-3.76	14.14
3	28	23.55	4.45	19.80
5	42	33.13	8.87	78.68
13	64	71.45	-7.45	55.50
8	50	47.5	2.5	6.25
16	90	85.82	4.18	17.47
11	58	61.87	-3.87	14.98
1	8	13.97	-5.97	35.64
9	54	52.29	1.71	2.92
				SSE = 245.38

Low variability

- In general, independent variables need some variability in order to be good predictors in a model.
- For instance, an underrepresented category in a variable (for example 195 non-smokers versus 5 smokers) is, in most cases, a good reason to remove the variable from the model. Although sometimes these variables can still be useful, for instance, when the outcome is lung cancer, even a highly skewed *smoking* variable may still be a good predictor.
- Variability should be considered together with the importance of the variable to decide whether to exclude it or not from the model.

- Importing the libraries
- Importing the dataset
- Splitting the dataset into the Training set and Test set
- Training the Simple Linear Regression model on the Training set
- Predicting the Test set results
- Visualising the Training set results
- Visualising the Test set results
- Making a single prediction (for example the salary of an employee with 12 years of experience)
- Getting the final linear regression equation with the values of the coefficients

Importing the libraries

- import numpy as np
- import matplotlib.pyplot as plt
- import pandas as pd

Importing the dataset

- `dataset = pd.read_csv('Salary_Data.csv')`
- `X = dataset.iloc[:, :-1].values`
- `y = dataset.iloc[:, -1].values`

YearsExperience	Salary
1.1	39343
1.3	46205
1.5	37731
2	43525
2.2	39891
2.9	56642
3	60150

Total -30 Rows...

Splitting the dataset into the Training set and Test set

- `from sklearn.model_selection import train_test_split`
- `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)`

Training the Simple Linear Regression model on the Training set

- `from sklearn.linear_model import LinearRegression`
- `regressor = LinearRegression()`
- `regressor.fit(X_train, y_train)`

Predicting the Test set results

- `y_pred = regressor.predict(X_test)`

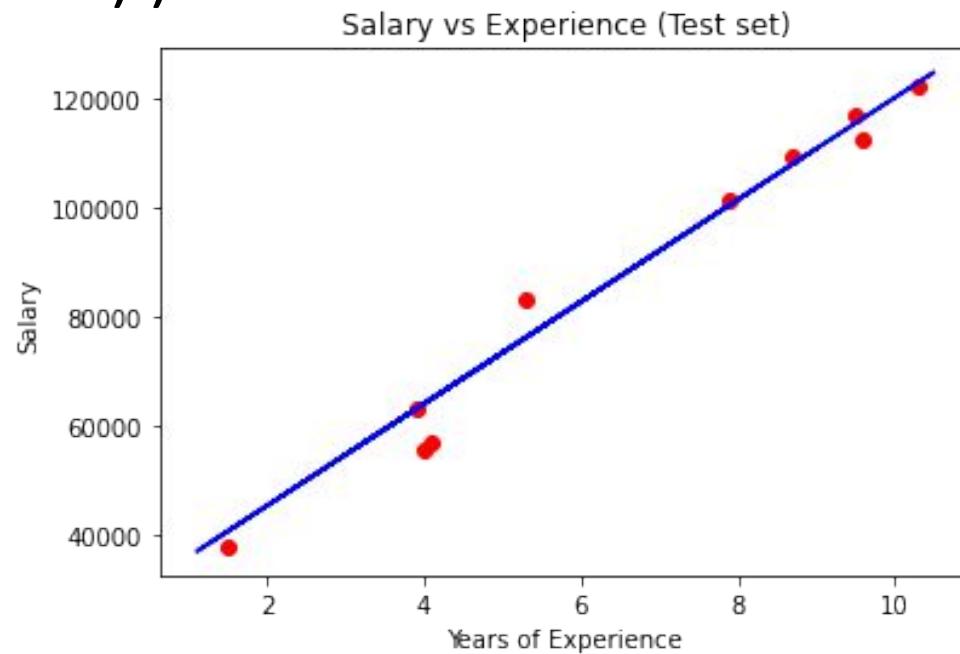
Visualizing the Training set results

- `plt.scatter(X_train, y_train, color = 'red')`
- `plt.plot(X_train, regressor.predict(X_train), color = 'blue')`
- `plt.title('Salary vs Experience (Training set)')`
- `plt.xlabel('Years of Experience')`
- `plt.ylabel('Salary')`
- `plt.show()`



Visualising the Test set results

- `plt.scatter(X_test, y_test, color = 'red')`
- `plt.plot(X_test, regressor.predict(X_test), color = 'blue')`
- `plt.title('Salary vs Experience (Test set)')`
- `plt.xlabel('Years of Experience')`
- `plt.ylabel('Salary')`
- `plt.show()`



Making a single prediction (for example the salary of an employee with 12 years of experience)

- `print(regressor.predict([[12]]))`
- output
 - [138967.5015615]
- Note:
 - 12- scalar
 - [12]- 1D Array
 - [[12]]- 2D Array

Getting the final linear regression equation with the values of the coefficients

- `print(regressor.coef_)`
- `print(regressor.intercept_)`
- Output
 - [9345.94244312]
 - 26816.192244031183
- Thus,
 - $\text{Salary} = 9345.94 \times \text{YearsExperience} + 26816.19$
- Note: `coef_` and `intercept_` are attributes of `LinearRegression`.

Multiple Linear Regression

Multiple Linear Regression

- $Y = b_0 + b_1 * x_1 + b_2 * x_2 + \dots + b_n * x_n$

OR

$$y = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2 + \theta_3 * x_3 + \dots$$

- We can also bring in a term x_0 , such that its value is always 1
- This allows you to write the equation in a further compact form:

$$y = \sum_{i=0}^n (\theta_i * x_i)$$

- This equation is also called a hypothesis function (or just hypothesis) and is represented by h_0 .
- Now, the need is to find the appropriate values of coefficients θ .
- Once that is done, it would be possible to predict y .
- There are two popular methods for computing θ : Normal Equation Method and Gradient Descent Method.

Normal Method

- You can think of the hypothesis being expressed as a matrix equation

$$X * \theta = Y$$

- Where X is matrix of dimensions $(m, n + 1)$, m rows (one row per data point), and $n + 1$ columns (first column with implicit feature value of 1 and then n columns – one per feature)
- θ is a column matrix of dimension $(n + 1, 1)$, having values θ_0 through θ_n
- Y is another column matrix of dimension $(m, 1)$, having values Y_1 through Y_m , representing the observed values for the m data points
- While you want the solution to above Eq. to be as accurate as possible, in reality, it might not be possible to find the value of θ such that the equation would be satisfied exactly for each of the m observations.
- Hence, the attempt is to find θ such that $X * \theta$ is as close to Y as possible.

$$X * \theta = Y_p$$

- You need to find θ such that Y_p is as close to Y as possible, i.e., $(Y - Y_p)$ should be as close to 0 as possible. And this should be evaluated on the basis of all the rows of data.
- Sometimes, a value of θ which improves the error for a given row could deteriorate the error for another row.
- So, you want to consider the total (or, average) errors across all the rows.
- When you consider the errors for all the rows, large positive error for a row could potentially be cancelled by large negative values on another row.
- To overcome this problem, you want to consider the square of an error, which is always positive, and hence, there is no chance of errors from different rows being cancelled by each other.
- Since Y, Y_p are both column matrixes, $Y - Y_p$ is also a column matrix, and squaring each number (to get rid of negative values) can be thought of as $(Y - Y_p)^T(Y - Y_p)$ where $(Y - Y_p)^T$ is transpose of matrix $(Y - Y_p)$.
- This is equivalent to squaring of each element of column matrix $(Y - Y_p)$.
- So, now, the requirement is to minimize $(Y - Y_p)^T(Y - Y_p)$

- The term that you want to minimize is called cost function or the loss function and is often represented by $J(\theta)$.

$$J(\theta) = (Y - X\theta)^T(Y - X\theta)$$

- Differential calculus says that for any curve, at its minimum value, the slope of the curve will be 0.
- Using matrix differential calculus, the slope of the $J(\theta)$, i.e., taking a derivative with respect to θ , will be $2X^T X\theta - 2X^T Y$
- Equating this slope to 0, you will get

$$2X^T X\theta - 2X^T Y = 0$$

$$X^T X\theta = X^T Y$$

$$\theta = (X^T X)^{-1} X^T Y$$

- So, given matrices X and Y for a large number of sample set, it is possible to determine the value of column matrix θ that will provide you with a way to predict a new value of y for a new set of features

Disadvantage of Normal Equation Method

- Normal Equation Method has one major disadvantage that needs matrix inversion ($X^T X$).
- Not all matrices can be inverted, and even when invertible, very large matrices can take a long compute time to invert.
- In such cases, we use another method called Gradient Descent.
- A general rule of thumb could be, if you have the $X^T X$ to be larger than 1000×1000 matrix, you may go with Gradient Descent method.
- Some of the reasons for matrices not being invertible could be:
 - Some of the features are directly related, and thus, some columns in X are redundant
 - In that case we need to get rid of redundant columns.
 - There are more variables than instances. This anyways means a non-unique solution.
 - This should be fixed by having many more instances, compared to the number of features that you have in your model.

Gradient Descent Method

- You know that predicted value $y_p = h_o = \theta_0 * x_0 + \theta_1 * x_1 + \dots + \theta_n * x_n$, and the error for any given dataset is equal to $y_p - y$.
- Since you have m data points, the total error over these m data points should be considered.
- And that total error is given by:

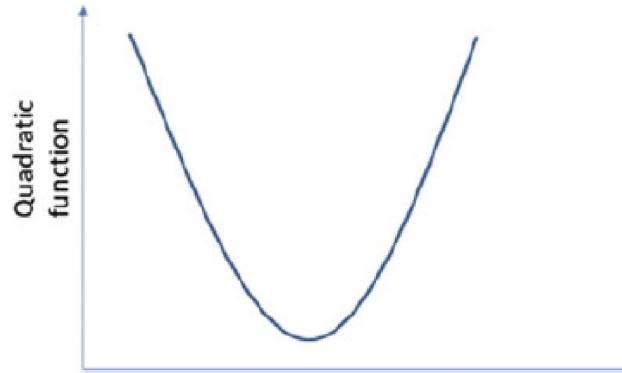
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h(x^i) - y^i)^2$$

- where
- $h(x^i)$ represents the predicted value y_p for the i th dataset
- $(h(x^i) - y^i)$ represents the error between the predicted value y_p and the actual observed value y for the i th dataset
- m is the number of data points in the dataset (such as historical data used in modeling)
- $J(\theta)$ is the total error term of the model for all the datasets

Gradient Descent Method

- And the division by $2m$ is for getting the average.
- Strictly speaking, you should be dividing by m for getting the average.
- However, we will be using a scaling by an arbitrary constant anyways.
- And so, you can divide (or, multiply) by any constant.
- Division by 2 is for ease in some further mathematical formulation.

- Since equation for the cost function is quadratic, its curve would look something like shown in Fig.



- This shape of the quadratic equation gives you a few very interesting properties.
- There is only one minimum value on this curve.
- The further you are from the local minima, the higher is the absolute value of the slope.
- The positive and negative value of the slope will also indicate whether you are to the right of the minima or to its left.

- The name Gradient Descent comes because the solution exploits these three properties of the quadratic curve.
- The solution will descend on this curve toward the minimum, using gradient (or, slope) direction and the value as a guidance.

Determine the Slope at Any Given Point

- the slope of $J(\theta)$ with respect to each of θ_j ($j = 0, 1, 2, \dots, n$).

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (h(x^i) - y^i) \cdot x_j^i \quad (\text{for } j = 0, 1, 2, \dots, n)$$

- if for any θ_j , the slope is positive, reduce the value of θ_j , and if the slope is negative, increase the value of θ_j .

Initial Value

- You can pick up any initial value for each of θ_j ($j = 0, 1, 2, \dots, n$).
- It just does not matter what value you choose.
- Often, people start with a choice of 0 for each of θ_j ($j = 0, 1, 2, \dots, n$).

Correction

- now, update each of these θ_j .
- The way to apply the correction is:

$$\theta_j = \theta_j - \alpha \cdot \frac{\delta J}{\delta \theta_j}$$

Learning Rate

- The slope (sign- direction) tells you whether to increase or decrease the θ_j .
- However, by how much? That “how much” is controlled by α , also called learning rate
- a smaller value of α will mean smaller steps, and hence it will take longer to reach the desired point.
- On the other hand, a larger value of α would mean larger steps. While the obvious advantage is that it will reach toward the solution faster, there is always a risk that it might overshoot the solution
- Thus, you need to choose a good value of α .
- Unfortunately, there is no good algorithmic way of finding the best value of α .
- You need to try various combinations and monitor if the solution continues to converge and how fast does it converge.
- So, try values of α in the range of 0.001 through 1, with various values in between, say 0.003, 0.01, 0.03, 0.1, 0.3, etc.
- Effectively, steps in multiple of ~ 3 . If in doubt, choose a lower, rather than higher value.

- Another interesting point is you would want to take larger steps, when you are further away from the solution, so that you don't have to take too many steps.
- And you would want to take smaller steps, when you are closer to the solution, so that you don't overstep.
- There is no need to change α for achieving this different step-size.
- This effect can be achieved due to slope changing.
- Given our cost function, the slope will be higher when you are further from the desired solution, and thus, you will be taking bigger steps.
- And closer to the desired solution, the slope would be smaller, and hence, you will be taking smaller steps.

- This concept of α is providing a scaling factor for applying the correction to computed slope.
- Since there is anyways a final scaling applied to the slope, the initial arbitrary scaling of slope is not an issue, since an appropriate choice of α will nullify the arbitrarily scaled slope earlier.
- This is the reason why we had added an additional factor of 2 in the denominator, which made the expression for slope easier.

Convergence

- A solution is said to be converging when continued iterations of the computation bring the solution closer to desired value
- How do you know you are done and don't need to continue applying corrections any further?
- How do you know the solution is even converging?
- One good method is to keep monitoring the cost function $J(\theta)$.
- With each iteration, $J(\theta)$ should be decreasing.
- As long as $J(\theta)$ is decreasing, you know that you are descending on the cost curve, and hence, the solution is converging.
- On the other hand, if $J(\theta)$ is increasing, or even oscillating, that means either something is wrong in our algorithm/coding or the choice of α is too high and should be reduced.
- So, monitoring $J(\theta)$ can help you confirm that the solution is converging.

when to stop applying corrections

- Now, you need to decide when to stop applying corrections and feel satisfied with the solution at hand.
- When the change in $J(\theta)$ is very small, you know that you are close to the solution.
- The cost function curve has shown that closer to the minima, the slope is very small, and hence, a very small change in $J(\theta)$ per iteration means you are close to the minimum.
- Depending on how close to the minimum do you want to go, you can choose the appropriate criterion for stopping to apply further corrections.

Note

- While determining the criterion for achieving closure, consider the value chosen for α .
- Assume you have chosen a very small value of α .
- That means the correction applied for each iteration would be very small, even if you are very far from the solution.
- So, a very small change in $J(\theta)$ may satisfy the criterion for your convergence.
- In reality, it is not that the solution has converged, just that your choice of α was so small that the iteration caused a very small step and hence, small improvement.

Gradient descent

Gradient Descent

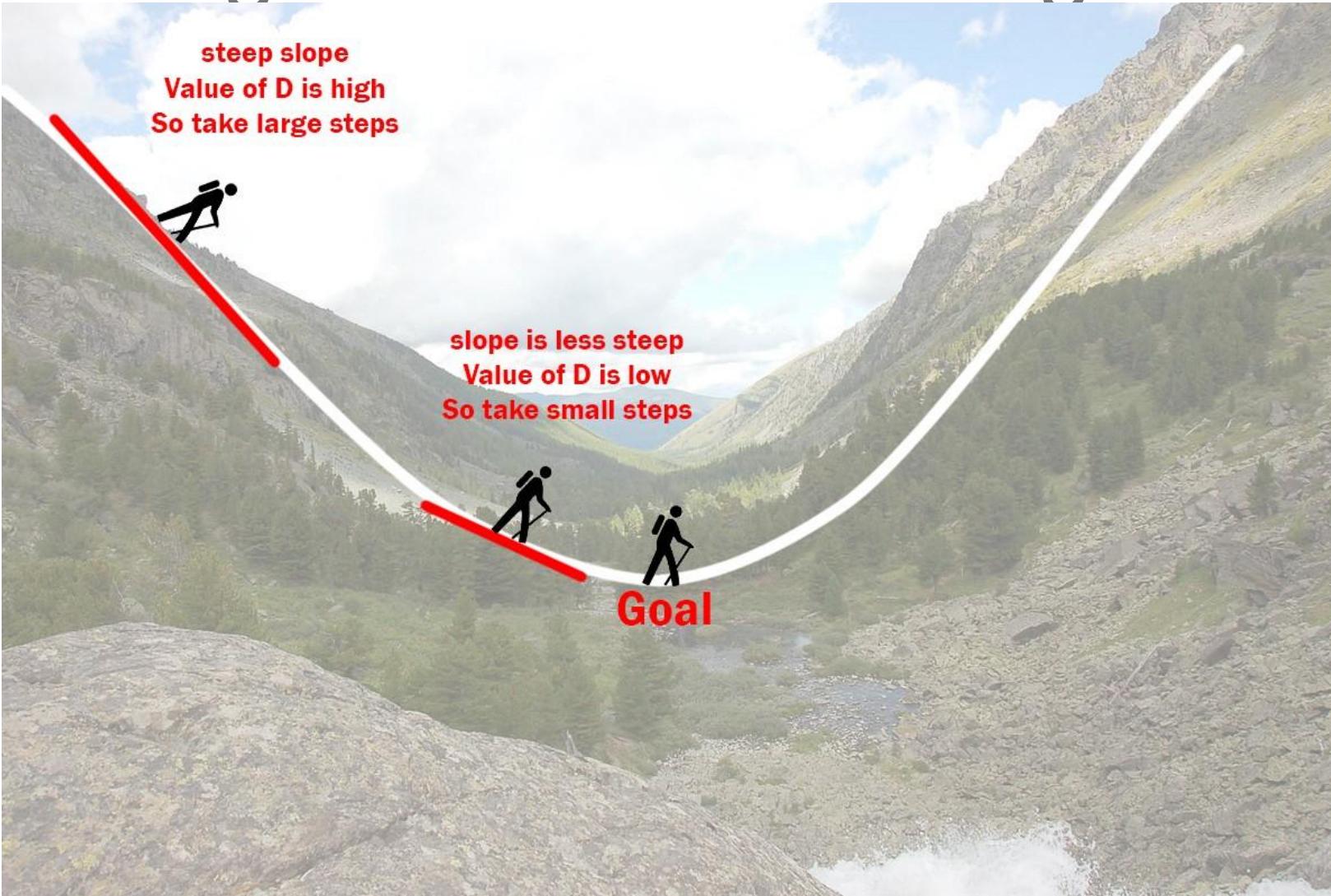
- is an iterative optimization algorithm to find the minimum of a function.
 - that function is our Loss Function
 - in this case the Mean Squared Error cost function.

Linear Regression

- $Y = mX + c$
- **Loss Function**
 - The loss is the error in our predicted value of m and c .
 - Our goal is to minimize this error to obtain the most accurate value of m and c .

$$E = \frac{1}{n} \sum_{i=0}^n (y_i - \bar{y}_i)^2$$

how the gradient descent algorithm works



Applying gradient descent step by step

- Initially let $m = 0$ and $c = 0$. Let L be our learning rate. This controls how much the value of m changes with each step. L could be a small value like 0.0001 for good accuracy.
- Calculate the partial derivative of the loss function with respect to m ,
 - plug in the current values of x , y , m and c in it to obtain the derivative value D .

$$D_m = \frac{1}{n} \sum_{i=0}^n 2(y_i - (mx_i + c))(-x_i)$$

$$D_m = \frac{-2}{n} \sum_{i=0}^n x_i(y_i - \bar{y}_i)$$

- Similarly lets find the partial derivative with respect to \mathbf{c} , D_c :

$$D_c = \frac{-2}{n} \sum_{i=0}^n (y_i - \bar{y}_i)$$

- Now we update the current value of \mathbf{m} and \mathbf{c} using the following equation

$$\mathbf{m} = \mathbf{m} - L \times D_m$$

$$\mathbf{c} = \mathbf{c} - L \times D_c$$

- We repeat this process until our loss function is a very small value or ideally 0 (which means 0 error or 100% accuracy).
 - The value of \mathbf{m} and \mathbf{c} that we are left with now will be the optimum values.

Linear Regression model

- The hypothesis for Linear Regression model

Hypothesis is noted as $h_{\theta}(x^{(i)})$ for a given input data $x^{(i)}$:

$$h_{\theta}(x^{(i)}) = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_j x_j$$

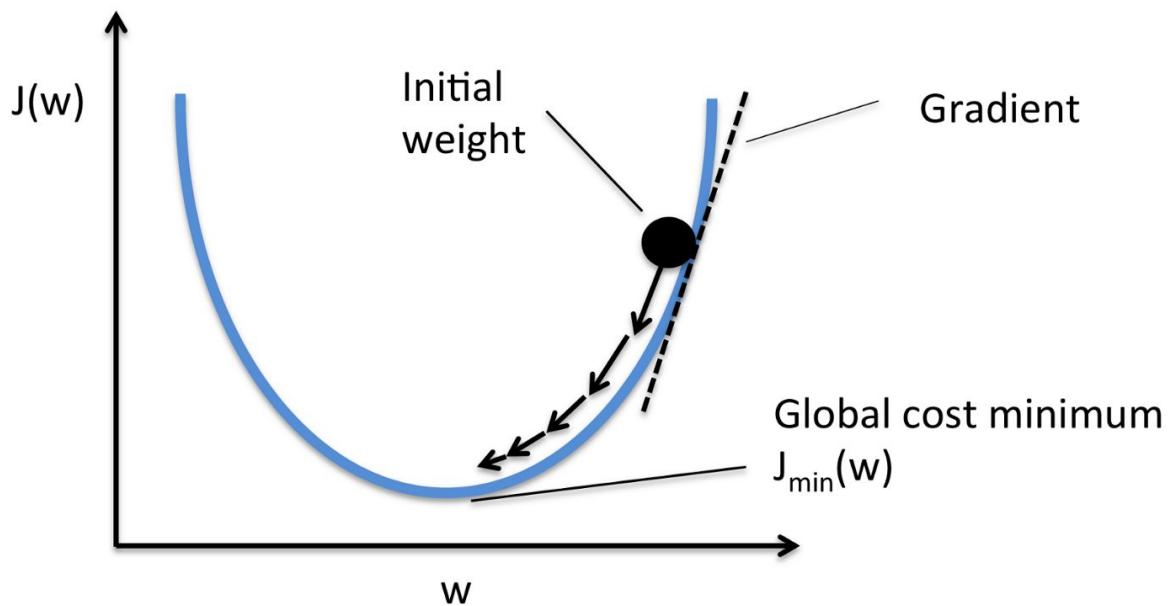
$x_0 = 1$, $j = \text{the number of features}$

- The most commonly used loss function for Linear Regression is
 - **Least Squared Error,**
 - and its cost function is also known as
 - **Mean Squared Error(MSE).**

$$\text{Least Squared Error} = \frac{1}{2}(h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$\text{Cost Function} = J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2, \quad m = \text{number of sample data}$$

- cost function is a parabola curve



- To minimize it, we need to find its vertex of the parabola
- One of the most popular programming solutions is **Gradient Descent**, to walk through the optimization process.
- Gradient Descent is pervasively applied in different models
- It takes steps based on a learning rate α , moving towards the vertex of the parabola
 - which is also known as the point with the slope equals to 0.

- So to get the slope, we take the derivative of cost function at each coefficient θ .
- Calculate the partial derivative of the loss function with respect to each coefficient θ

Derivative of cost function at θ_j :

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- Next, continuously update each θ with a good amount of iterations along with the observation of reduction in cost function until it reaches a horizontal line.
- In reality, it is impossible for the slope to reach an exact value of 0 due to the chosen step size, but it can be close to 0 as much as possible depending on hyperparameters.

$$\theta'_0 = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta'_1 = \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

.....

$$\theta'_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Stochastic gradient descent

- each training instance is shown to the model one at a time.
- The model makes a prediction for a training instance,
- the error is calculated and
- the model is updated in order to reduce the error for the next prediction.
- here gradient is the error for the model on the training data.

Types of Gradient Descent Algorithms

- 1. Batch Gradient Descent
- 2. Stochastic Gradient Descent
- 3. Mini-Batch Gradient Descent

Batch Gradient Descent

- In the batch gradient descent, to calculate the gradient of the cost function,
 - we need to sum all training examples for each steps
- If we have 3 millions samples (m training examples) then the gradient descent algorithm should sum 3 millions samples for every epoch
- To move a single step, we have to calculate each with 3 million times
- Batch Gradient Descent is not good fit for large datasets

$$\theta'_0 = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta'_1 = \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

.....

$$\theta'_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Stochastic Gradient Descent (SGD)

- In stochastic Gradient Descent,
 - we use one example or one training sample at each iteration instead of using whole dataset to sum all for every steps
- SGD is widely used for larger dataset trainings
- Computationally faster and can be trained in parallel
- Need to randomly shuffle the training examples before calculating it

Mini-Batch Gradient Descent

- It is similar like SGD, it uses n samples instead of 1 at each iteration.

Multiple Linear Regression

Confounding vars

- A confounding variable is a variable that is related to both the independent variable and the dependent variable, and can potentially affect the observed relationship between them.
- For example
 - in a study examining the effect of a new drug on blood pressure,
 - age may be a confounding variable because it is related to both the drug and blood pressure.
 - To study the effect of exercise on weight loss,
 - calorie may be a confounding var
 - If we omit calorie intake, the model might incorrectly attribute the effect of calorie intake (a confounder) to exercise.
 - In reality, both exercise and calorie intake influence weight loss, but calorie intake might be the stronger factor.
 - Not considering calorie intake can make it seem like exercise has a greater effect on weight loss than it actually does.

Mitigation

- **Include potential confounders in the regression model**
 - This helps isolate the effect of the independent variable by controlling for other variables that might be influencing the outcome.
- **Use techniques like stratification or matching**
 - In more complex scenarios, techniques like stratification (analyzing groups separately) or matching (comparing similar groups) can help reduce confounding.

Stratification

- divide the data into subgroups, or "strata," based on the levels of a confounding variable.
- examine relationship between the independent and dependent variable within each stratum
- generalize the relationship by combining results

Stratification Example

- to study the effect of a new drug (independent variable) on blood pressure reduction (dependent variable) and
 - you suspect that gender is a confounder because
 - it influences both drug response and baseline blood pressure levels.
- **Step 1:** Stratify the data into two groups based on gender (e.g., male and female).
- **Step 2:** Analyze the effect of the drug on blood pressure separately within the male group and the female group.
- **Step 3:** Compare the results to see if the effect of the drug is consistent across genders or if gender was indeed a confounder.

Controlling for confounding variables

- we can include covariates, such as age, as independent variables in the MLR model to control for their effects on the dependent variable.
- By controlling for confounding variables, we can isolate the effect of the independent variable on the dependent variable and determine if there is a causal relationship.
- $Y = \beta_0 + \beta_1 X + \beta_2 Z + \epsilon$
- Y = outcome (heart disease risk)
- X = main predictor of interest (exercise)
- Z = potential confounder (age)
- β_1 estimates the effect of X on Y, **holding Z constant**.

mediator variable

- A **mediator** is a variable that lies **on the causal pathway** between a predictor (independent variable) and an outcome (dependent variable).
 - It helps *explain how or why* the predictor affects the outcome.
 - So the predictor influences the mediator, and the mediator in turn influences the outcome.
- Example of a mediator
- to study the effect of **education level** on **income**.
 - Higher education → improves **skills** (mediator) → leads to higher income. Here, **skills** is a mediator between education and income.
- If you control for **skills** in your regression, the estimated effect of education on income will shrink,

Another example of mediator var

- **Smoking → Lung Damage → Lung Cancer**
Here, lung damage **is the mediator**.
- Smoking does not directly “cause” cancer as much as it causes lung damage, which then leads to cancer.

What happens when you control for the mediator (lung damage)?

- $\text{Cancer} = \beta_0 + \beta_1(\text{Smoking}) + \beta_2(\text{Lung Damage}) + \varepsilon$
- Once you include **lung damage**, the model asks:
 - For two people with the *same level of lung damage*, but different smoking habits, how different is their risk of cancer?
 - But if lung damage is the *main way smoking leads to cancer*, then among people with equal lung damage, smoking won't explain much anymore
 - So it looks like “removing the effect”

Mediator vs. Confounder

- **Confounder:** A variable that affects both the predictor and the outcome, creating a spurious association.
 - You *should* control for it.
- **Mediator:** A variable that transmits part of the effect from predictor to outcome.
 - Whether to control depends on your research question:
 - If you want the **total effect** → do **not** control.
 - If you want the **direct effect** (excluding mediation) → do control.

Which Variables Should You Include In A Regression Model?

- When building a linear or logistic regression model, you should consider including:
 1. Variables that are already proven in the literature to be related to the outcome
 2. Variables that can either be considered the cause of the exposure (independent var), the outcome (response var), or both
 3. Interaction terms of variables that have large main effects
- Interaction terms:
 - $\text{Income} = \beta_0 + \beta_1(\text{Education}) + \beta_2(\text{Gender}) + \beta_3(\text{Education} \times \text{Gender}) + \varepsilon$
 - If $\beta_3 \neq 0$, then education doesn't have the same effect for men and women.

Which Variables Should You not Include In A Regression Model?

- However, you should watch out for:
 1. Variables that have a large number of missing values or low variability
 2. Variables that are highly correlated with other predictors in the model (causing a collinearity problem)
 3. Variables that are not linearly related to the outcome (in case you're running a linear regression)

Which var to control for (include in the model)

- If the variable is a **confounder** (affects both predictor and outcome)
 - you *should* control.
- If it's a **mediator** (lies on the causal path)
 - controlling might “block” the effect you want to measure.
- If it's irrelevant
 - controlling adds unnecessary noise.

Collinearity

- A situation in which two or more independent variables in a regression model are highly correlated with each other.
 - which means they share some degree of linear relationship.
- Makes it difficult to determine the individual effect of each variable on the dependent variable because their effects are not easily separable.
- Example
 - if you're predicting house prices and include both the number of bedrooms and the square footage of the house as predictors,
 - these two variables might be highly correlated (bigger houses tend to have more bedrooms).

How to detect collinearity?

- Not all collinearity problems can be detected by inspection of the correlation matrix.
 - This is because a correlation matrix shows correlation between PAIRS of variables.
- Sometimes collinearity exists between 3 or more variables — multicollinearity,
 - this can be detected by looking at the VIF (Variance Inflation Factor).
- In general, a $VIF > 10$ indicates a multicollinearity issue.
 - Not hard and fast: sometimes multicollinearity can be present even with a $VIF < 5$
- Once collinearity is detected: Solution:
 - the simplest solution is to keep the most important of the variables and drop the rest.

Impact on Regression Model

- **Unstable Coefficient Estimates**
 - The coefficients of the correlated variables may become highly sensitive to small changes in the model, leading to large variances in the estimated coefficients.
- **Interpretation Challenges**
 - It becomes difficult to interpret the individual effect of each variable because their contributions to the model are not independent.
- **Reduced Model Reliability**
 - High collinearity can make the model less reliable and less generalizable to new data.

Detection and Mitigation

- **Variance Inflation Factor (VIF)**
 - A common method to detect multicollinearity is to calculate the VIF for each predictor.
 - A VIF value above 10 is often considered indicative of high multicollinearity.
- **Removing or Combining Variables**
 - You can address collinearity by removing one of the correlated variables, combining them into a single variable, or using techniques like principal component analysis (PCA).
- **Regularization Techniques**

Variance Inflation Factor (VIF)

- is a statistical measure used to detect the presence and severity of multicollinearity in a regression model.
- **VIF Calculation:** For each independent variable in a regression model, VIF is calculated as:
 - R^2 gives how well your model is fitted to the data
 - where R_j^2 is the coefficient of determination (R-squared) obtained by regressing the j-th independent variable on all the other independent variables in the model.
 - treat the j-th independent variable (predictor) as the dependent variable in a new regression model, and all the other independent variables from the original model are treated as the predictors in this new regression.
- for $x_1 = b_0 + b_1 * x_2 + b_2 * x_3$
 - R_1^2 represents the proportion of the variance in X_1 that can be explained by X_2 and X_3 .

VIF values: interpretation

VIF = 1: The independent variable is not correlated with any other variables. There is no multicollinearity.

1 < VIF ≤ 5: There is moderate correlation, but it is generally not severe enough to be problematic.

VIF > 5: indicates a high correlation, and there might be a problem with multicollinearity.

VIF > 10: Indicates serious multicollinearity, which likely requires corrective action as the estimates of the regression coefficients are unreliable.

Avoiding collinearity when selecting predictors

- When 2 or more independent variables in a model are highly correlated we say that we have a collinearity problem.
- Collinearity is not a binary issue: either we have it or we don't.
 - the stronger the correlation between inputs of the regression model, the more serious the problem is.
- When variables are highly correlated, it becomes hard to correctly estimate the contribution of each single one of them.

Including variables that are not linearly related to the outcome

- This part only applies to linear regression, where we assume that the relationship between predictors and outcome should be a straight line.
- Don't underestimate the importance of this assumption, because when violated, all of the linear model's output is USELESS.

What if you ended up with a large number of predictors?

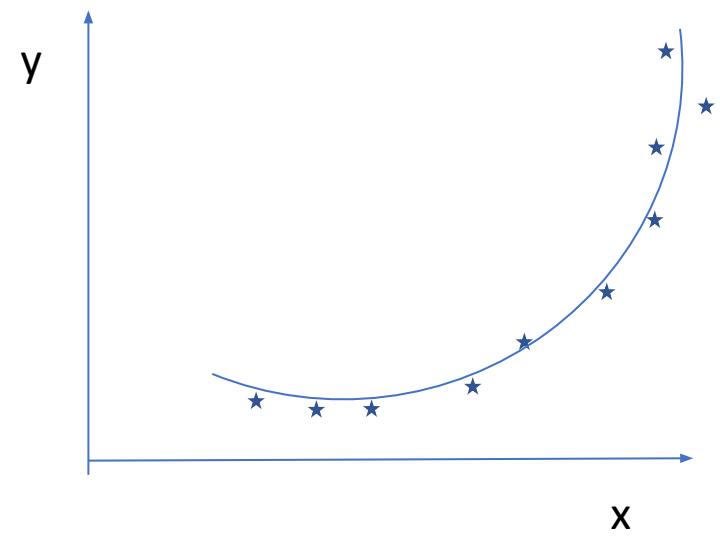
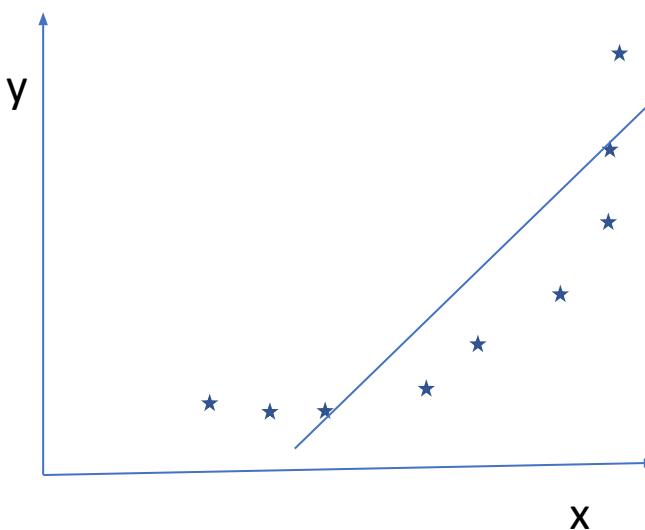
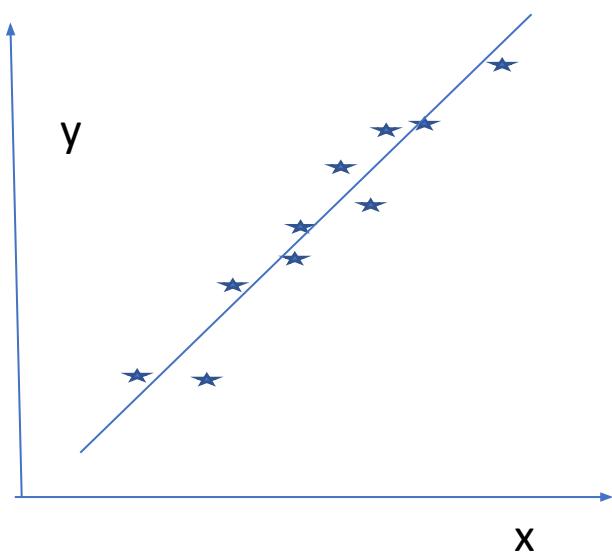
- When fitting a linear regression model, the number of observations should be at least 15 times larger than the number of predictors in the model.
- For a logistic regression, the count of the smallest group in the outcome variable should be at least 15 times the number of predictors.
- Otherwise, the model will not be generalizable — its out of sample accuracy will be low because of overfitting.

Reduce the number of candidate predictors by

- **Grouping similar variables:** Either by using domain knowledge, for example replacing all diseases that affect heart muscles, valves, rhythm and blood vessels with one variable — cardiovascular disease. Or based on statistical methods like PCA (Principle Components Analysis) which summarizes the large initial set of variables by replacing them with 2 or 3 principle components that maintain the largest possible variability in the data. Grouping variables has the downside of getting a coefficient for the combined variables only, therefore losing the details on the effect of each.
- **Running a LASSO regression:** LASSO is a regularized regression model that shrinks the coefficients of unimportant predictors to zero, thus performing automatic variable selection. However, all coefficients will be shrunk and the effect of each predictor will appear lower than it really is.
- **Using stepwise variable selection:** Despite its popularity, stepwise regression has a lot of limitations that you should be aware of before using it.
- **Using univariate variable selection:** Another popular and highly criticized method is to run a hypothesis test on each candidate variable, then in the final model only include those that had a p-value < 0.2 for example. This approach is just a variant of stepwise selection and therefore inherits the same problems of the stepwise method.
- **Selecting variables that have the largest variance:** In cases where you have thousands of predictors, it is common to reduce their number by only selecting those with the highest overall variability as these are more likely to be predictive of the outcome. The advantage of using such method to select predictors regardless to the outcome variable is that it does not bias the regression results.

Polynomial Regression

Data



Polynomial Linear Regression

- When simple linear regression would not be relevant
 - Non-linear dataset
- We can use a linear model to fit nonlinear data
 - Add powers of each feature as new features, then train a linear model on this extended set of features
- This technique is called as polynomial regression
- $Y = b_0 + b_1 x + b_2 x^2 + \dots + b_n x^n$

- Why Linear?
 - In regards to the coefficients not the independent var
 - Thus it is a special case of multiple linear regression
- Usecase
 - To describe how diseases spread across population

In case of multiple features

- In this case, polynomial regression is capable of finding relationships between features.
- This is made possible by the fact that it adds all combinations of features up to the given degree.
 - Relationship we can get from the coefficient of these newly added features (having combination of different features).
- For e.g.
 - if there were two features a and b
 - And select 3 degree polynomial
 - It would not only add a^2 , a^3 , b^2 and b^3 but also the combinations ab , a^2b and ab^2 .
- It transforms an array containing n features into an array containing $(n+d)!/d!n!$ features
 - Combinatorial explosion of the number of features.

Physical significance of coefficients

- In case of SLR or MLR each coefficient represents the contribution or effect of that particular term on the model's prediction
- Coefficients of squared terms (e.g., x_1^2, x_2^2)
 - These terms model **nonlinear relationships** between the feature and the target
 - x_1^2 means that the feature x_1 affects the target in a **quadratic fashion**.
 - The corresponding coefficient quantifies the **curvature** of the relationship between the feature and the target.
 - A positive coefficient implies a **concave upward** (U-shaped) relationship
 - A negative coefficient implies a **concave downward** (inverted-U) shape.

Physical significance of coefficients

- Cross-product terms (e.g., x_1x_2)
- These terms model **interactions** between two features
 - indicates the effect of one feature on the target depends on the value of the other feature.
 - The coefficient of this term represents the **magnitude of interaction**:
 - If the coefficient is positive
 - increasing both x_1 and x_2 together will increase the target more than the sum of their individual effects.
 - If the coefficient is negative
 - increasing both x_1 and x_2 together will reduce the target more than the sum of their individual effects.

Physical significance of coefficients

- Higher-order interaction terms (e.g., $x_1^2x_2$)
 - These represent **more complex interactions**, showing how the squared value of one feature interacts with another.

Polynomial Regression is Non-local

- Changes to the polynomial can **reshape the curve** significantly, affecting predictions across the entire feature space.
- Changes to the polynomial
 - Changes in the coefficients
 - Changes in the Degree of the Polynomial
 - Influence of Data Points
- This is different from methods like **k-nearest neighbors (KNN)**, which are purely local and only consider nearby data points for predictions.

Polynomial Regression is Non-local

- Changes to the polynomial
 - Changes in the coefficients
 - changing the value of a coefficient in a high-degree polynomial affects the shape of the curve not just near the data points but also far away from them.
 - If β_2 (the coefficient of x_2) increases, the curvature of the quadratic term becomes steeper, affecting the model's behavior across the entire input range.
 - Changing β_1 (the linear term) would alter the slope of the entire regression line.

Polynomial Regression is Non-local

- Changes to the polynomial
 - Changes in the Degree of the Polynomial
 - The **degree** of the polynomial refers to the highest power of the feature variable(s) in the regression model
 - **Increasing the degree** allows the model to fit more complex curves, potentially capturing more intricate patterns in the data.
 - For example:
 - A quadratic model (x^2) can fit parabolic shapes, while a cubic model (x^3) can fit S-shaped curves.
 - A higher-degree polynomial can fit curves that oscillate more, bending and twisting around data points.

Polynomial Regression is Non-local

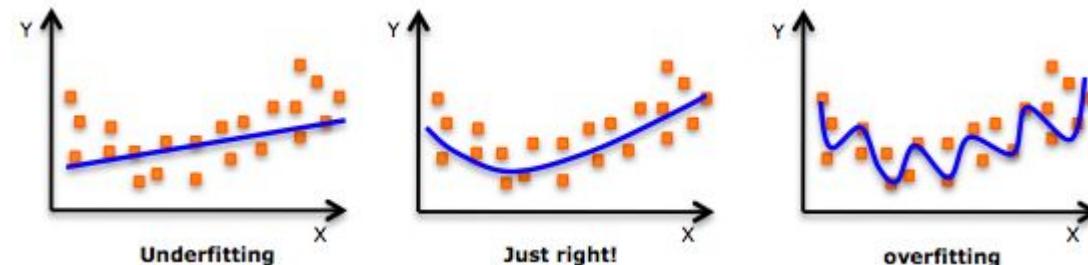
- Influence of Data Points
 - As polynomial regression fits a single polynomial curve across all data
 - thus a single data point or outlier can influence the shape of the regression curve far away from that point.

Pros and Cons

- works on any size of dataset
- works very well on non linear problems
- need to choose the right polynomial degree for a good bias/variance tradeoff
 - polynomial regression has a tendency to drastically over-fit, even on this simple one dimensional data set.
- it is inherently non-local, i.e., changing the value of Y at one point in the training set can affect the fit of the polynomial for data points that are very far away.
 - Hence, to avoid the use of high degree polynomial on the whole dataset, we can substitute it with many different small degree polynomial functions.

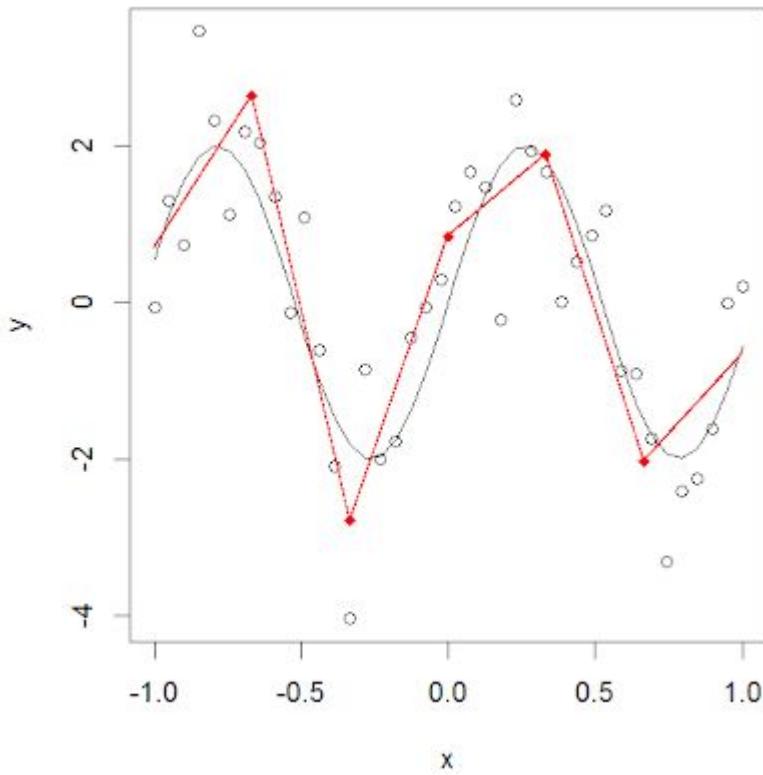
Regression Splines

- we noticed both it's advantages and limitations of Linear Regression.
- It assumed a linear relationship between the dependent and independent variables, which was rarely the case in reality.
- As an improvement over this model, we have Polynomial Regression which generate better results (most of the time).
- But using Polynomial Regression on datasets with high variability chances to result in over-fitting.
- It is inherently non-local, i.e., changing the value of Y at one point in the training set can affect the fit of the polynomial for data points that are very far away.



Solution-Regression Splines

- It uses a combination of linear/polynomial functions to fit the data.
- instead of building one model for the entire dataset, divides the dataset into multiple bins and fits each bin with a separate model.
- one of the most important non linear regression techniques.
 - A polynomial (like ax^2+bx+c) is a smooth curve that spans the **entire domain** of the input variable. When you fit such a polynomial, the predicted curve is determined by all data points together, which imposed a global structure on the dataset not just a local neighborhood.
 - To overcome this, we can divide the distribution of the data into separate portions and fit linear or low degree polynomial functions on each of these portions.



- The points where the division occurs are called Knots.
- Functions which we can use for modelling each piece/bin are known as Piecewise functions.

Piecewise Step Functions

- One of the most common piecewise functions is a Step function.
- Step function is a function which remains constant within the interval.
- We can fit individual step functions to each of the divided portions in order to avoid imposing a global structure.
- Here we break the range of X into bins, and fit a different constant in each bin.

- we create cut points C_1, C_2, \dots, C_K in the range of X , and then construct $K + 1$ new variables.

$$\begin{aligned}
 C_0(X) &= I(X < c_1), \\
 C_1(X) &= I(c_1 \leq X < c_2), \\
 C_2(X) &= I(c_2 \leq X < c_3), \\
 &\vdots \\
 C_{K-1}(X) &= I(c_{K-1} \leq X < c_K), \\
 C_K(X) &= I(c_K \leq X),
 \end{aligned}$$

- where $I()$ is an indicator function that returns a 1 if the condition is true and returns a 0 otherwise. For example, $I(c_K \leq X)$ equals 1 if $c_K \leq X$, otherwise it equals 0. For a given value of X , at most only one of C_1, C_2, \dots, C_K can be non-zero, as X can only lie in any one of the bins.
- **Binned regression does not create continuous functions of the predictor, so in most cases we would expect no relationship between the input and output.**

Piecewise Polynomials

- Instead of fitting a constant function over different bins across the range of X, piecewise polynomial regression involves fitting separate low-degree polynomials over different regions of X.
- As we use lower degrees of polynomials, we don't observe high oscillations of the curve around the data.
- For example, a piecewise quadratic polynomial works by fitting a quadratic regression equation:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2$$

- where the coefficients β_0 , β_1 and β_2 differ in different parts of the range of X.

- A piecewise cubic polynomial, with a single knot at a point c , takes the below form:

$$y_i = \begin{cases} \beta_{01} + \beta_{11}x_i + \beta_{21}x_i^2 + \beta_{31}x_i^3 + \epsilon_i & \text{if } x_i < c; \\ \beta_{02} + \beta_{12}x_i + \beta_{22}x_i^2 + \beta_{32}x_i^3 + \epsilon_i & \text{if } x_i \geq c. \end{cases}$$

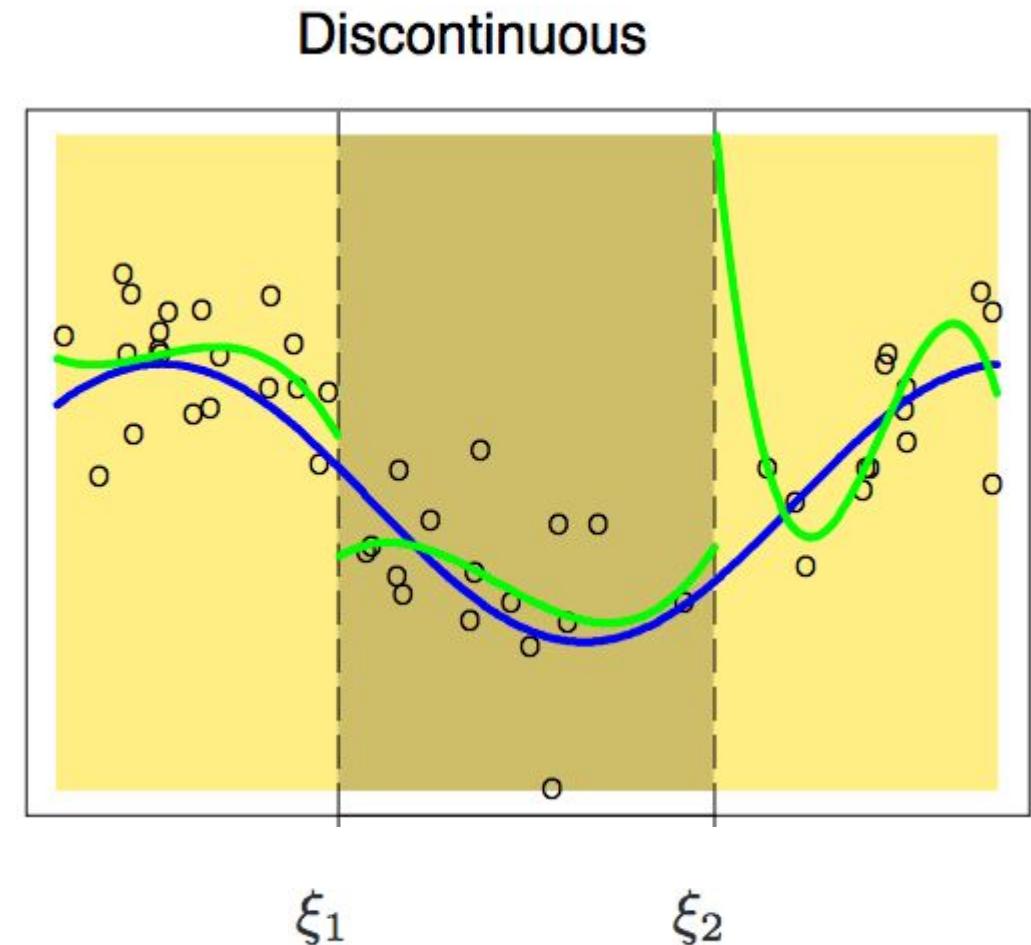
- In other words, we fit two different polynomial functions to the data: one on the subset of the observations with $x_i < c$, and one on the subset of the observations with $x_i \geq c$.
- The first polynomial function has coefficients $\beta_{01}, \beta_{11}, \beta_{21}, \beta_{31}$ and the second has coefficients $\beta_{02}, \beta_{12}, \beta_{22}, \beta_{32}$.
- **Each of these polynomial functions can be fit using the least squares error metric.**
- This family of polynomial functions has 8 degrees of freedom, 4 for each polynomial (as there are 4 variables).

- Using more knots leads to a more flexible piecewise polynomial, as we use different functions for every bin.
- These functions depend only on the distribution of data of that particular bin.
- In general, **if we place K different knots throughout the range of X, we will end up fitting K+1 different cubic polynomials.**
- We can use any low degree polynomial to fit these individual bins.
- For example, we can instead fit piecewise linear functions.
- In fact, **the stepwise functions used above are actually piecewise polynomials of degree 0.**

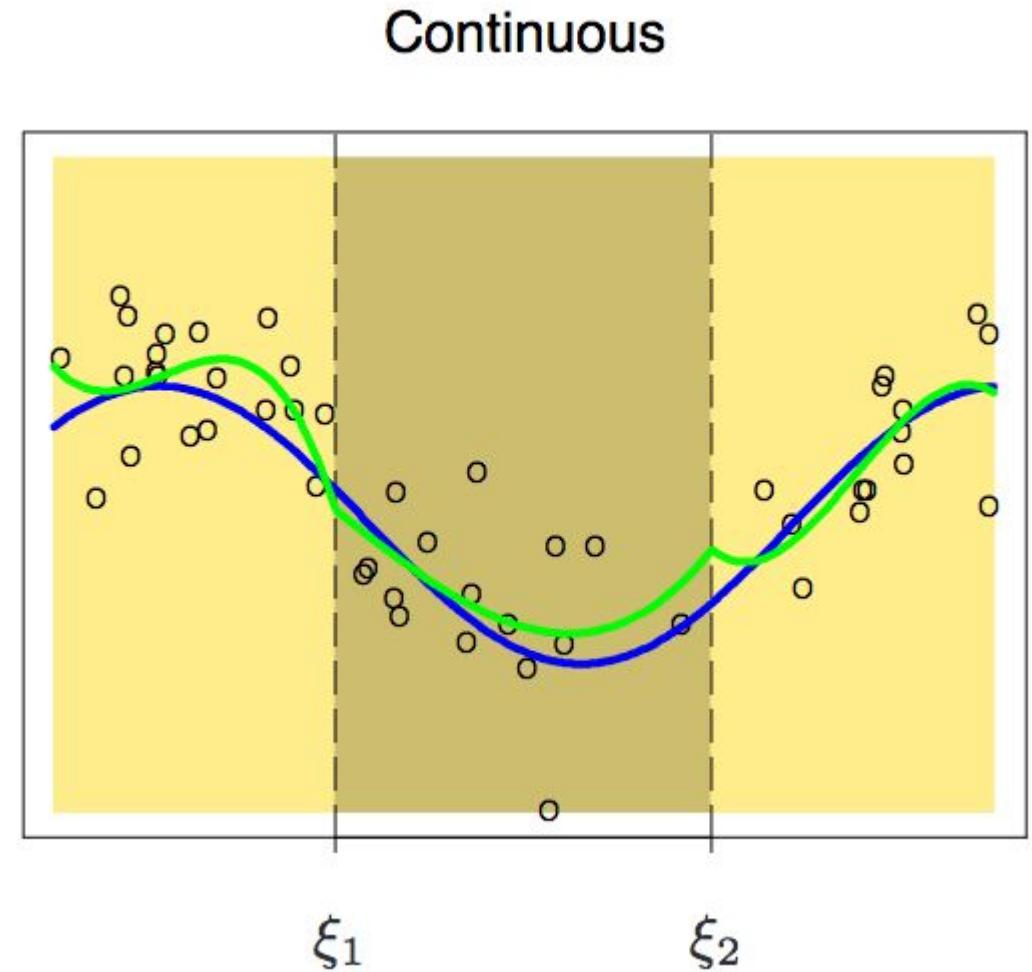
necessary conditions and constraints for forming piecewise polynomials

- Constraints and Splines

- We might encounter certain situations where the polynomials at either end of a knot are not continuous at the knot.
- Such a condition should be avoided because the family of polynomials as a whole should generate a unique output for every input.

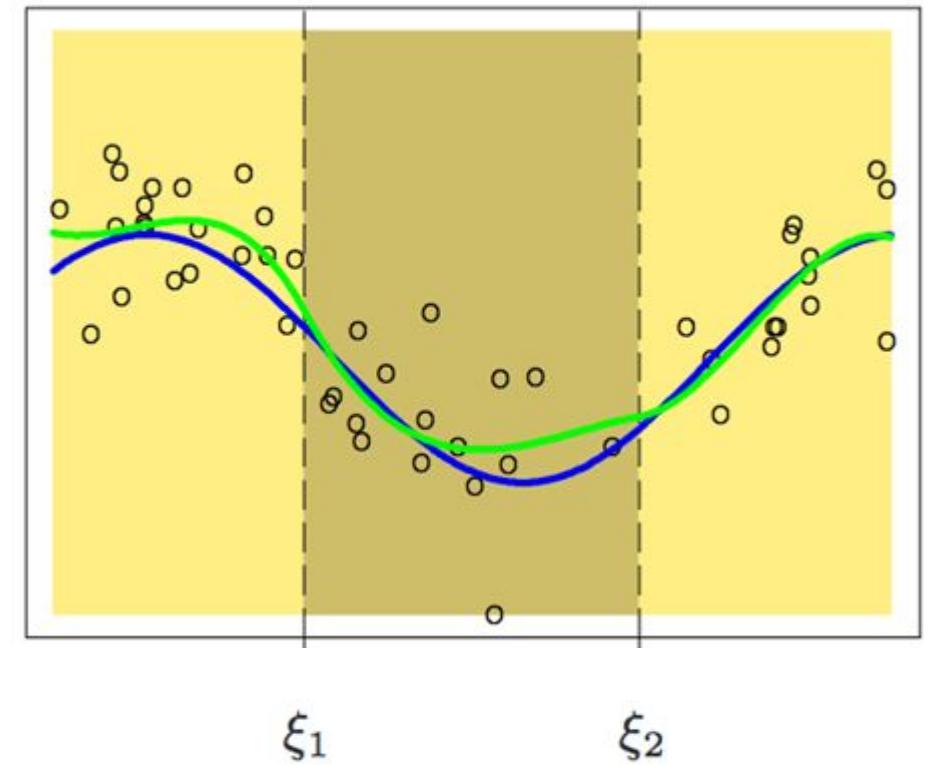


- We can see from the above image that it outputs two different values at the first knot.
- Thus, to avoid this, we should **add an extra constraint/condition that the polynomials on either side of a knot should be continuous at the knot.**
- Now after adding that constraint, we get a continuous family of polynomials.
- It looks like smoothness at the knots is still absent.



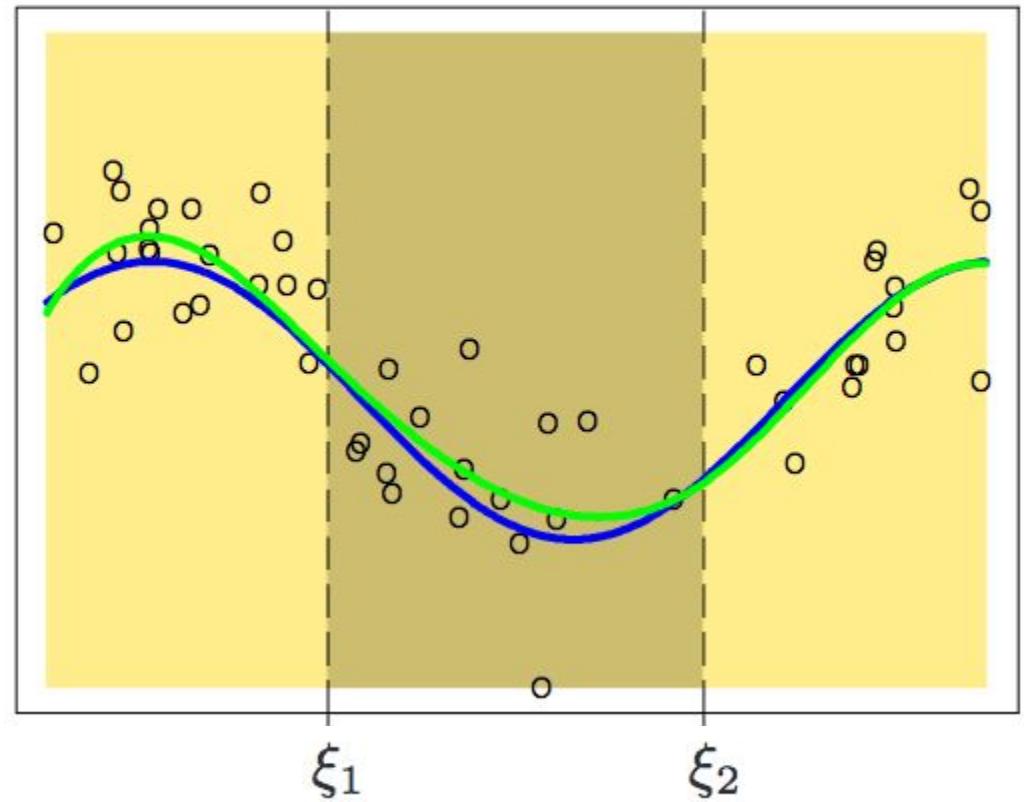
- So to smoothen the polynomials at the knots, we **add an extra constraint/condition: the first derivative of both the polynomials must be same.**
- One thing we should note: *Each constraint that we impose on the piecewise cubic polynomials effectively frees up one degree of freedom*, as we reduce the complexity of the resulting piecewise polynomial fit.
- Therefore, in the above plot, we are using only 10 degrees of freedom instead of 12.
- This plot uses 8 degrees of freedom instead of 12 as two constraints are imposed.

Continuous First Derivative



- Although the above plot looks better, there is still some scope for improvement.
- Now, **we will impose an extra constraint: that the double derivatives of both the polynomials at a knot must be same.**
- This plot seems perfect for our study. It uses 6 degrees of freedom instead of 12.
- **Such a piecewise polynomial of degree m with m-1 continuous derivatives is called a Spline.**
- Hence, we have constructed a Cubic Spline in the above plot.
- We can plot any degree of spline with m-1 continuous derivatives.

Continuous Second Derivative



Cubic and Natural Cubic Splines

- Cubic spline is a piecewise polynomial with a set of extra constraints (continuity, continuity of the first derivative, and continuity of the second derivative).
- In general, a cubic spline with K knots uses cubic spline with a total of $4 + K$ degrees of freedom.
- There is seldom any good reason to go beyond cubic-splines (unless one is interested in smooth derivatives).
- We know that the behavior of polynomials that are fit to the data tends to be erratic near the boundaries.
- Such variability can be dangerous.
- These problems are resembled by splines, too.
- The polynomials fit beyond the boundary knots behave even more wildly than the corresponding global polynomials in that region.
- **To smooth the polynomial beyond the boundary knots, we will use a special type of spline known as Natural Spline.**
- A natural cubic spline adds additional constraints, namely that the function is linear beyond the boundary knots. This constrains the cubic and quadratic parts there to 0, each reducing the degrees of freedom by 2. That's 2 degrees of freedom at each of the two ends of the curve, reducing $K+4$ to K .

Choosing the Number and Locations of the Knots

- When we fit a spline, where should we place the knots?
- One potential place would be the area of high variability, because in those regions the polynomial coefficients can change rapidly. Hence, one option is to place more knots in places where we feel the function might vary most rapidly, and to place fewer knots where it seems more stable.
- While this option can work well, in practice it is common to place knots in a uniform fashion. One way to do this is to specify the desired degrees of freedom, and then have the software automatically place the corresponding number of knots at uniform quantiles of the data.
- Another option is to try out different numbers of knots and see which produces the best looking curve.

- **A more objective approach is to use cross-validation.** With this method:
 - we remove a portion of the data,
 - fit a spline with a certain number of knots to the remaining data, and then,
 - use the spline to make predictions for the held-out portion.
- We repeat this process multiple times until each observation has been left out once, and then compute the overall cross-validated RMSE.
- This procedure can be repeated for different numbers of K knots.
- Then the value of K giving the smallest RMSE is chosen.

K-Nearest Neighbor

K-Nearest neighbor

- Instance based learning- also called lazy algorithm
 - When we get the training examples we do not process them and learn the model instead we just store the examples
 - And when we need to predict an instant that time we process
 - Algorithm does not come up with the model apriori rather when it gets the test instance it uses the stored instances in memory in order to find the possible model
- Non-linear model
- Very compute intensive
- default parameter for the number of neighbors k
 - 5

How does it do it?

- For a given new instance you find what is the closest instance and you find the y value of that closest instance and you guess this y values as the y - value of the test instance.
- How to find most similar instance/some neighbouring (most nearest) instances
 - We can use similarity metrics (or distance functions) depending on the type of data that we have
 - Euclidean distance, Cosine and other

1-nearest neighbour

- Training phase- save the examples
- Prediction time- we get the test instance (x_t) and find the training example (x_i, y_i) that is closest to test instance (x_t) and we predict the y_i as the output y_t

K-nearest neighbour

- Training phase- save the examples
- Prediction time- we get the test instance (x_t) and find k training example $(x_1, y_1), (x_2, y_2) \dots (x_k, y_k)$ that are closest to test instance (x_t) and we predict
 - the average of y_1, y_2, \dots, y_k as the estimate of y_t

improvements/issues of concern

- Weighting examples from the neighbourhood in case of $k > 1$
- How to measure closeness
 - Many distance functions
- How to find the closest points quickly
 - We must use some good data structure or method to store training examples so that we don't need to go through all the training examples, find the distance and come up with the nearest data point

How to prepare model

- 1. Choose the number K of neighbors
- 2. Take the K-nearest neighbors of the new data point, according to the Euclidean distance
- 3. For these K neighbors, calculate the average of y values
- 4. Assign the new data point to the calculated y value

Decision Tree

Decision Trees

- Classification trees
- Regression trees
- The goal is to create a model that predicts the value of a target variable by **learning simple decision rules** inferred from the data features.
- The deeper the tree, the more complex the decision rules and the fitter the model.
- Regression trees are more complex than classification trees
- Decision tree regression model is not best model to use on single feature dataset
 - Its more adapted to datasets with many features (high dimensional datasets)

- Classification
 - partition input space into *class regions*
- Regression
 - partition input space into *numeric prediction zones*
 - trying to approximate a continuous function

Example

X (feature)	Y (target for regression)	Class (target for classification)
1	2	A
2	3	A
3	10	B
4	12	B

Classification Tree

- Goal: predict Class (A or B)
 - a. If $X \leq 2 \rightarrow$ assign Class A
 - b. Else ($X > 2 \rightarrow$ assign Class B)
- The tree is just separating the input space into two regions (A vs B).
 - a. For $X=1.5 \rightarrow$ prediction = A
 - b. For $X=3.5 \rightarrow$ prediction = B
- No need to care about *how far apart* the values are within each region.

Regression Tree

- Goal: predict Y (numeric value)
 - a. If $X \leq 2 \rightarrow$ predict mean of Y in this region = $(2 + 3)/2 = 2.5$
 - b. If $X > 2 \rightarrow$ predict mean of Y in this region = $(10 + 12)/2 = 11$
- So the model has to output actual numbers (2.5 or 11).
 - a. For $X=1.5 \rightarrow$ prediction ≈ 2.5
 - b. For $X=3.5 \rightarrow$ prediction ≈ 11
- If we had more points, the regression tree would keep splitting to better approximate the continuous curve (e.g., going from $2 \rightarrow 3 \rightarrow 10 \rightarrow 12$).
- Unlike classification
 - it's not just about separating groups—
 - it's about matching the numeric *function* $Y=f(X)$

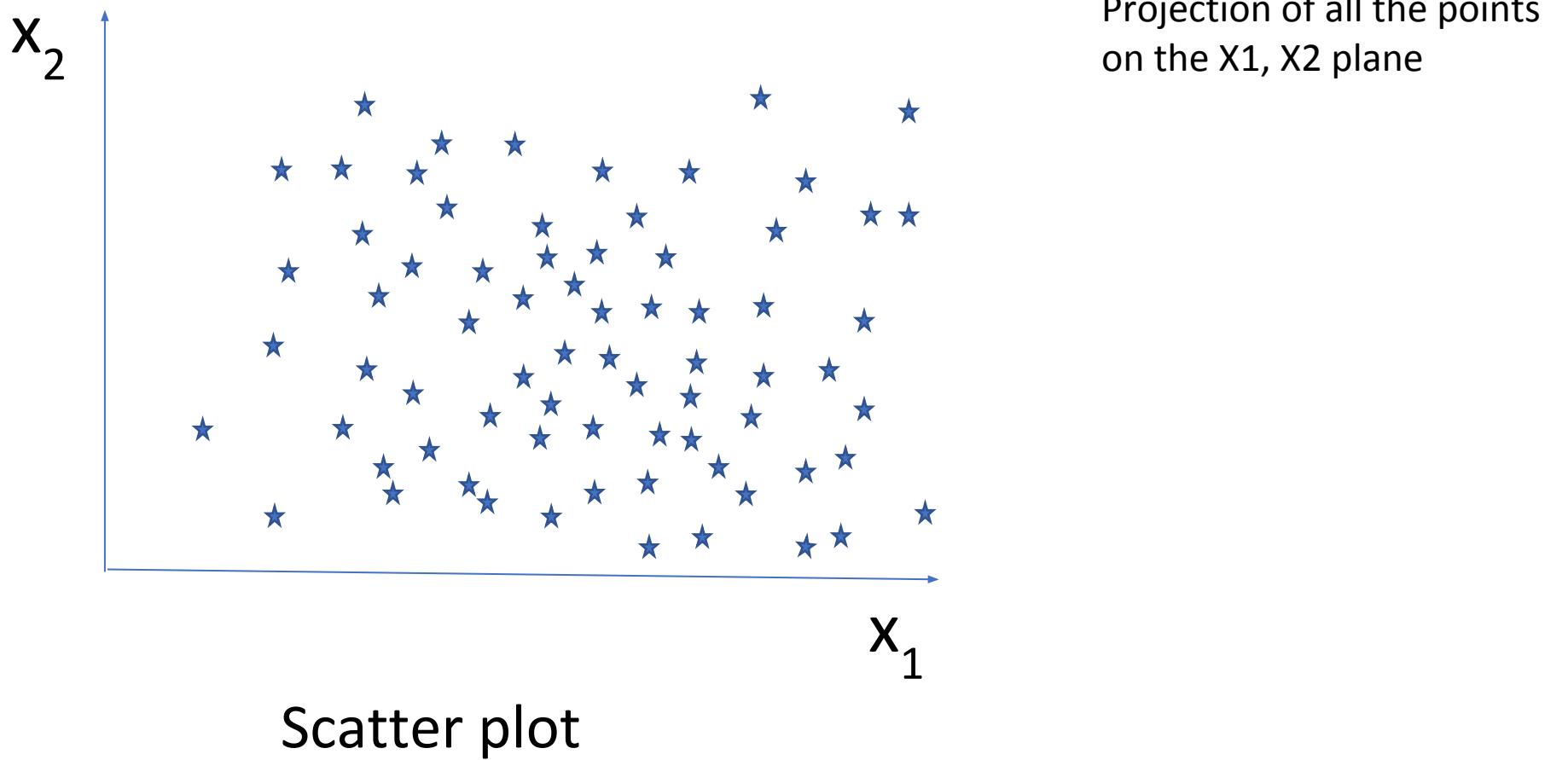
Pros and Cons

- Interpretability
 - no need for feature scaling
 - works on both linear / nonlinear problems
-
- Poor results on too small datasets, overfitting can easily occur

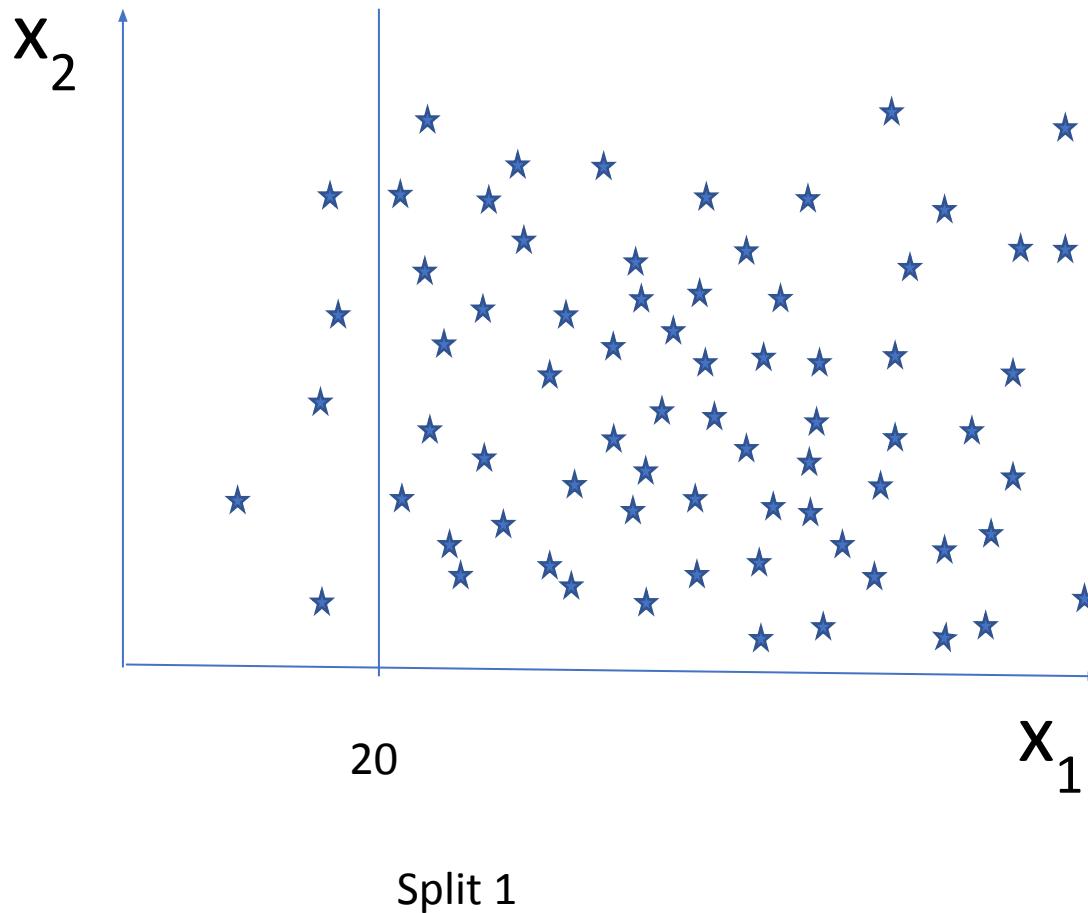
Decision Trees

- They were popular around 30 years ago but then more sophisticated methods replaced them
- Recently they were reborn with new upgrades
 - Additional methods that build on top of decision trees
 - Random forests, gradient boosting etc.
 - It is quite a simple method but it lies in the foundation of some of the more modern and powerful methods.

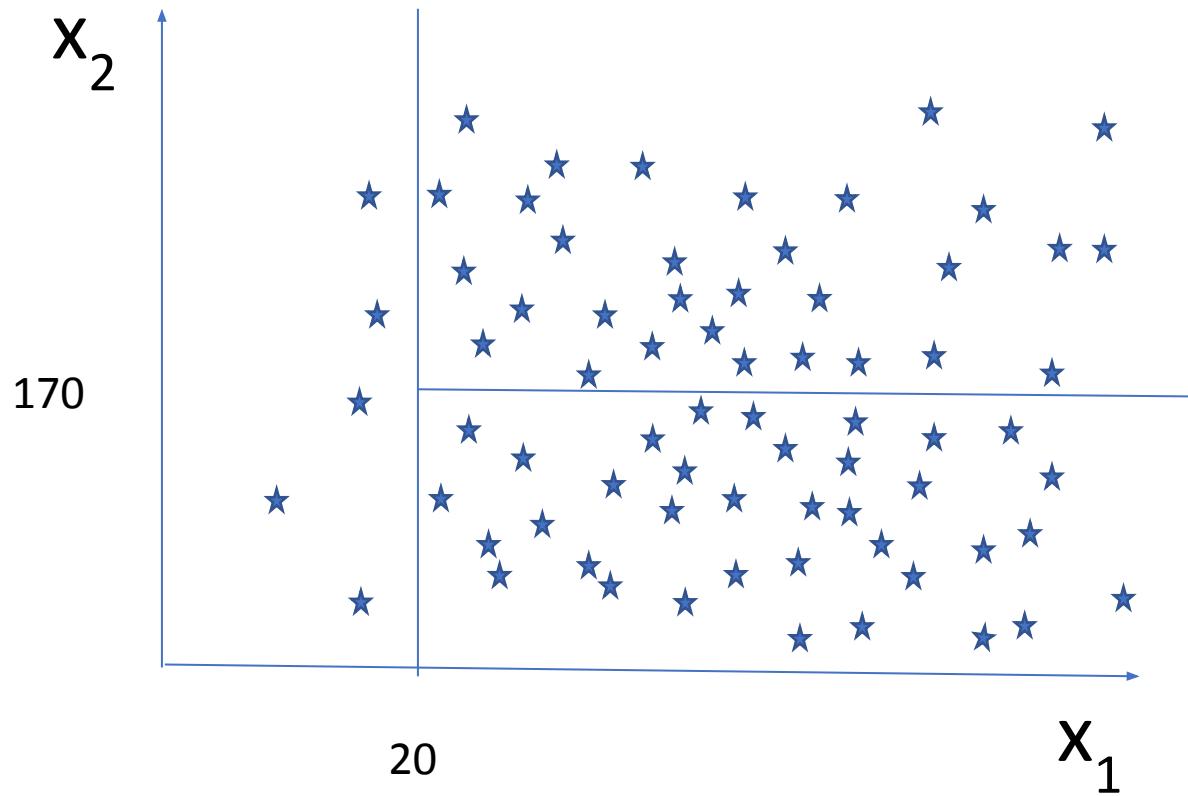
$x_1, x_2 \rightarrow$ independent vars, y-dependent var



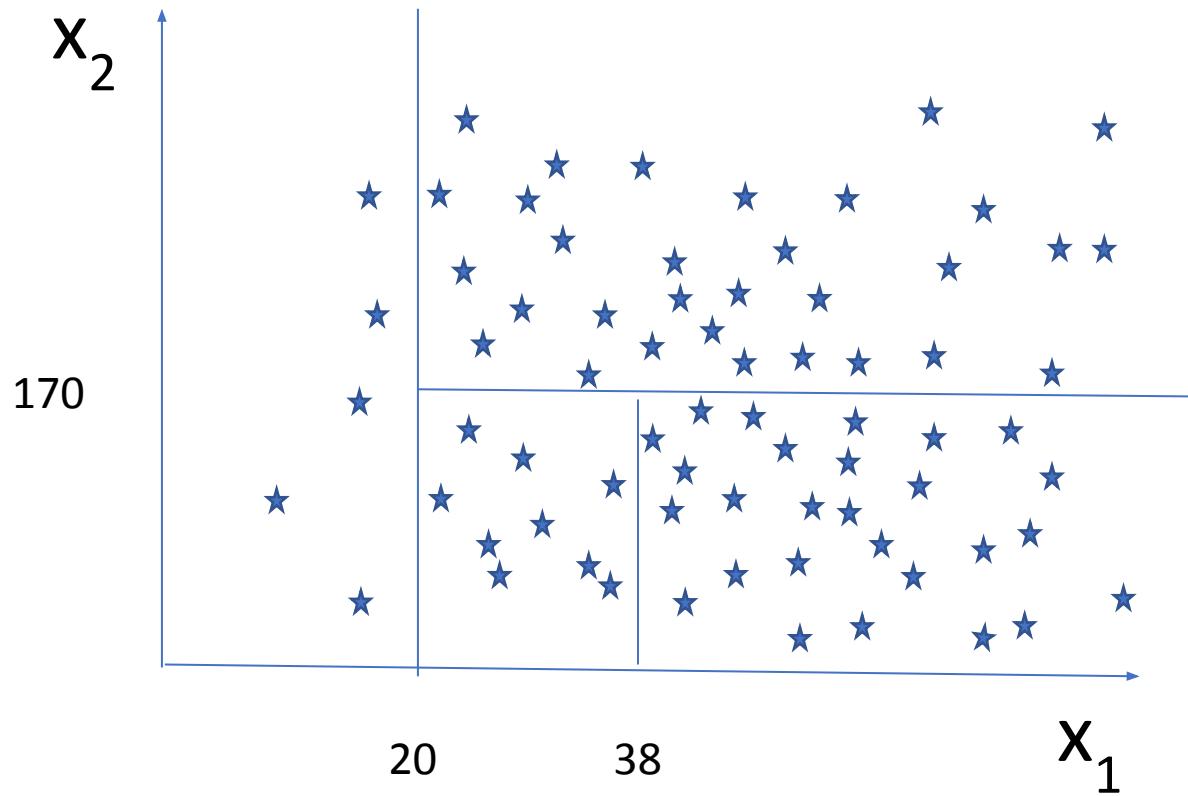
When we run Decision tree algorithm data will be split up into segments.



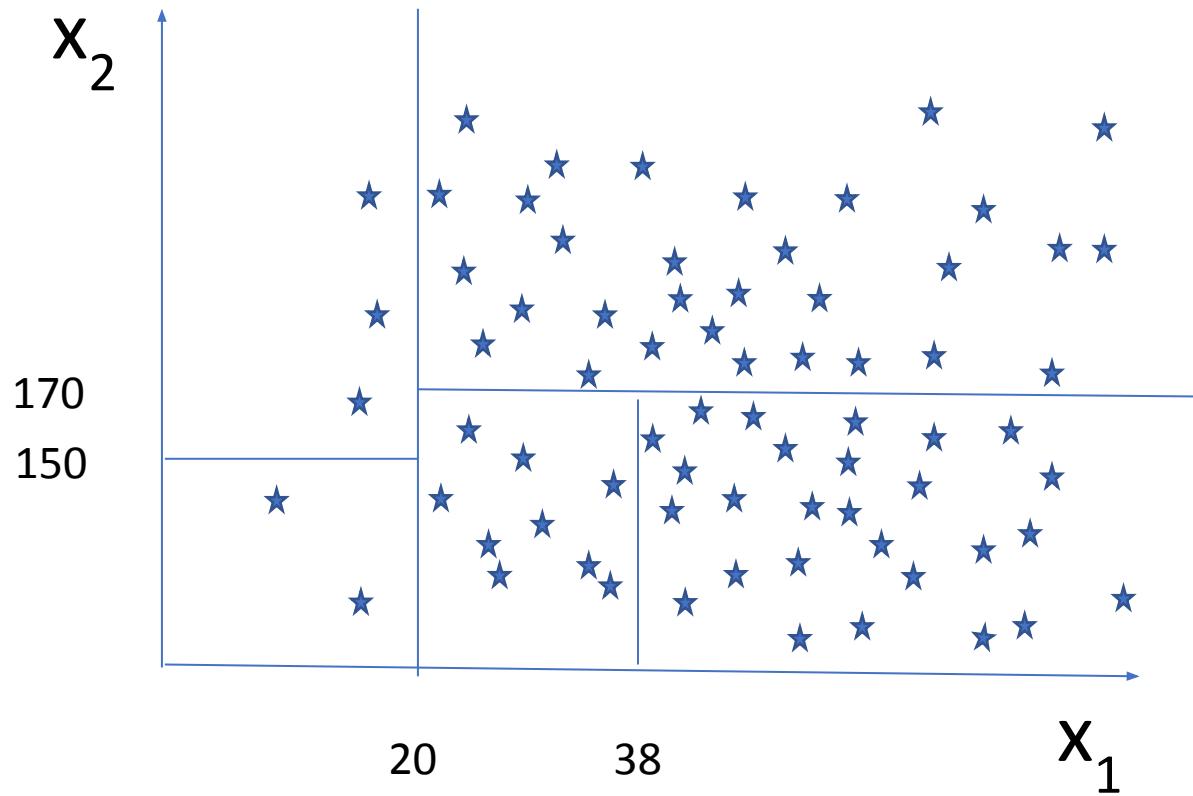
Split 2



Split 3



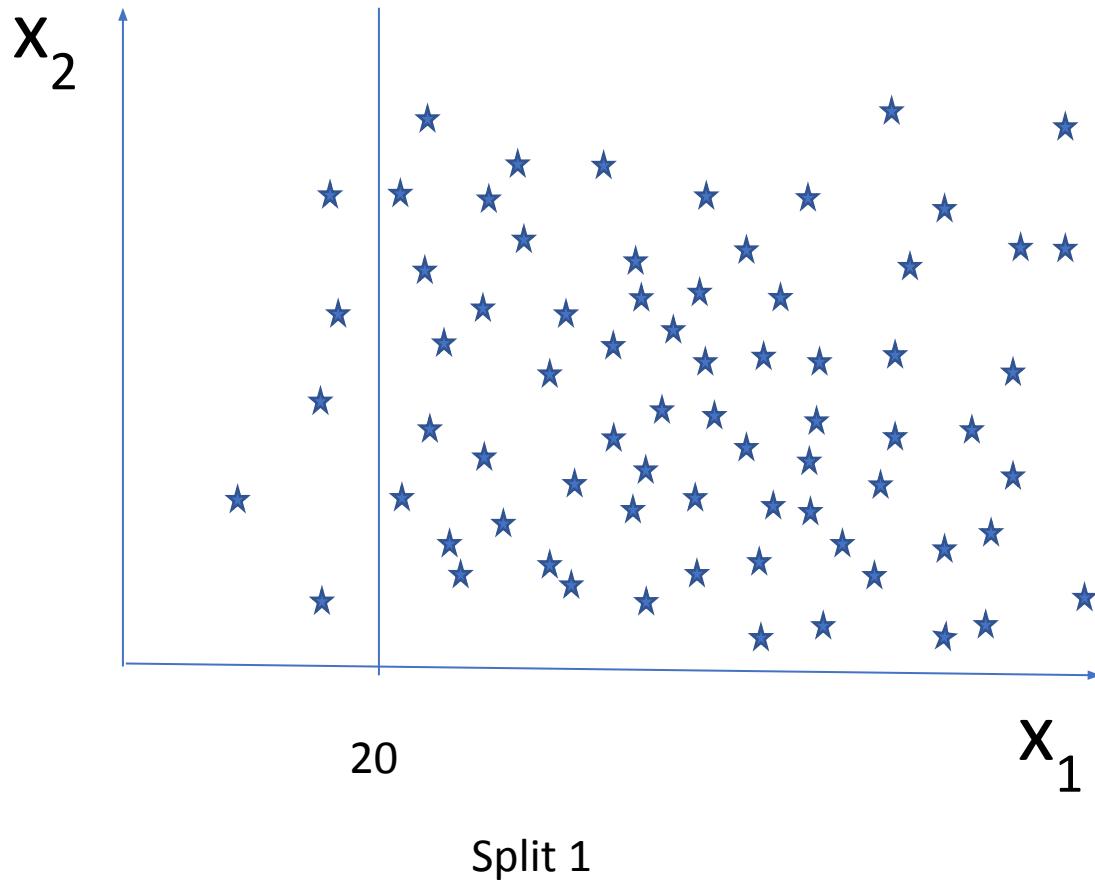
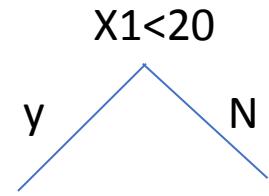
Split 4



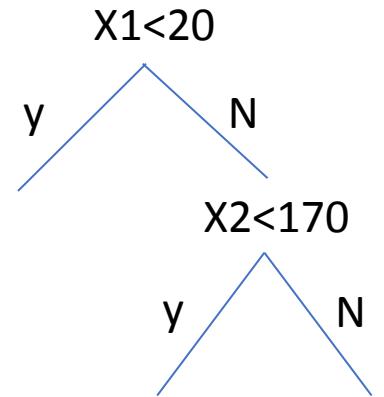
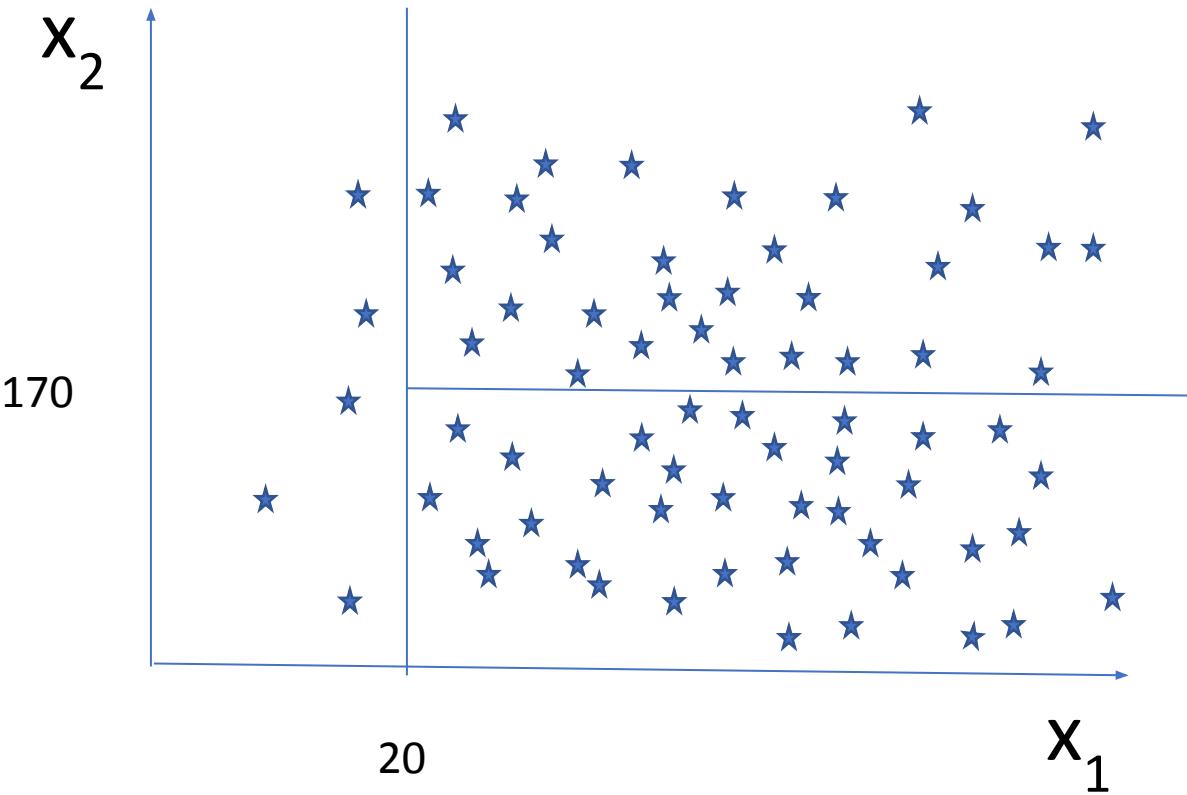
Algorithm

- It is finding the optimal splits of the datasets in to leaves
- Final leaves are called terminal leaves
- How and where split is conducted is determined by the algorithm
Information entropy
- Split increases the amount of information that we have about the points
- Algorithm stops when splitting can not add any information to our setup by splitting any leave
 - So algorithm stops if Less then 5% of total points in that leaf (different variations/options)

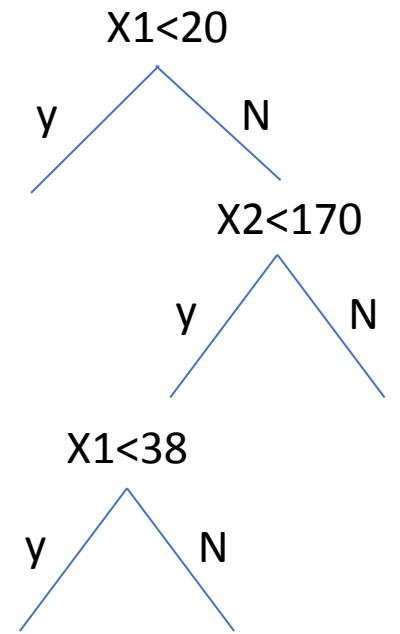
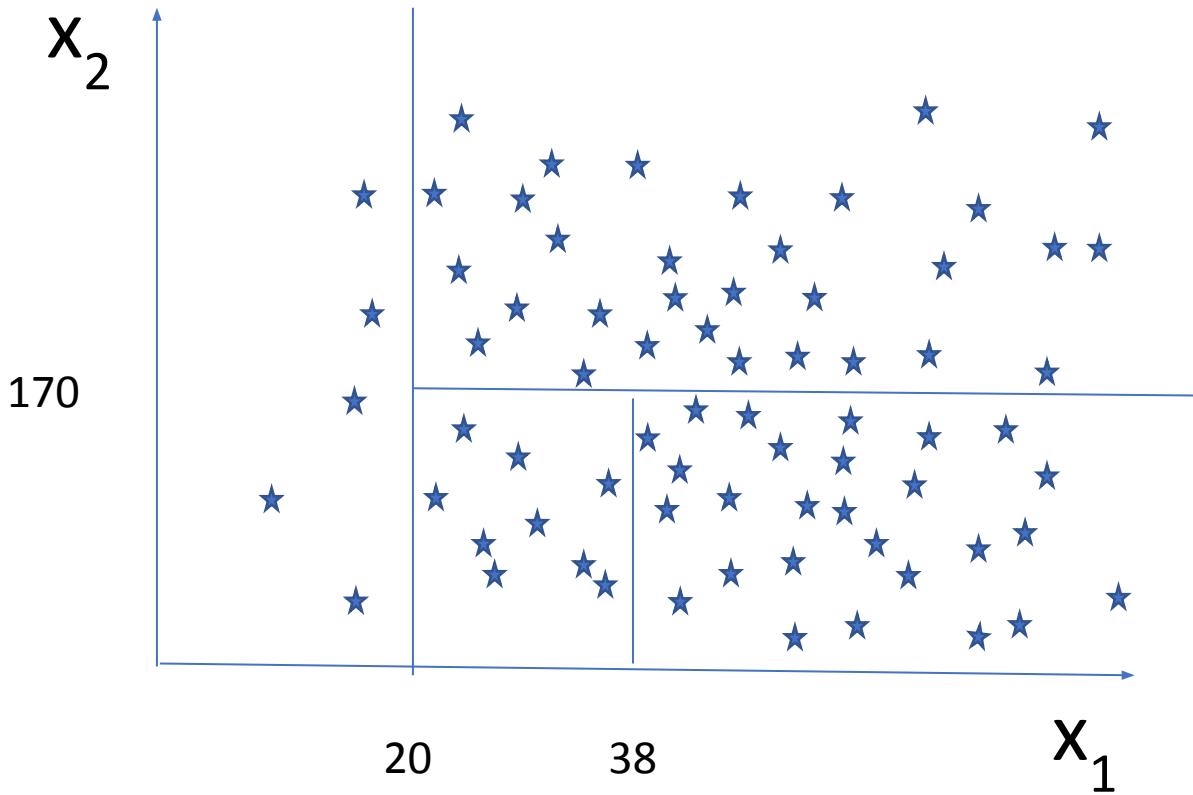
Creating decision tree



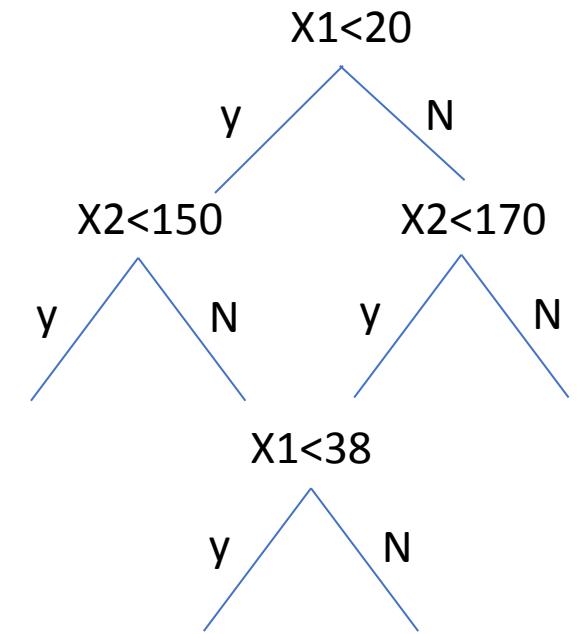
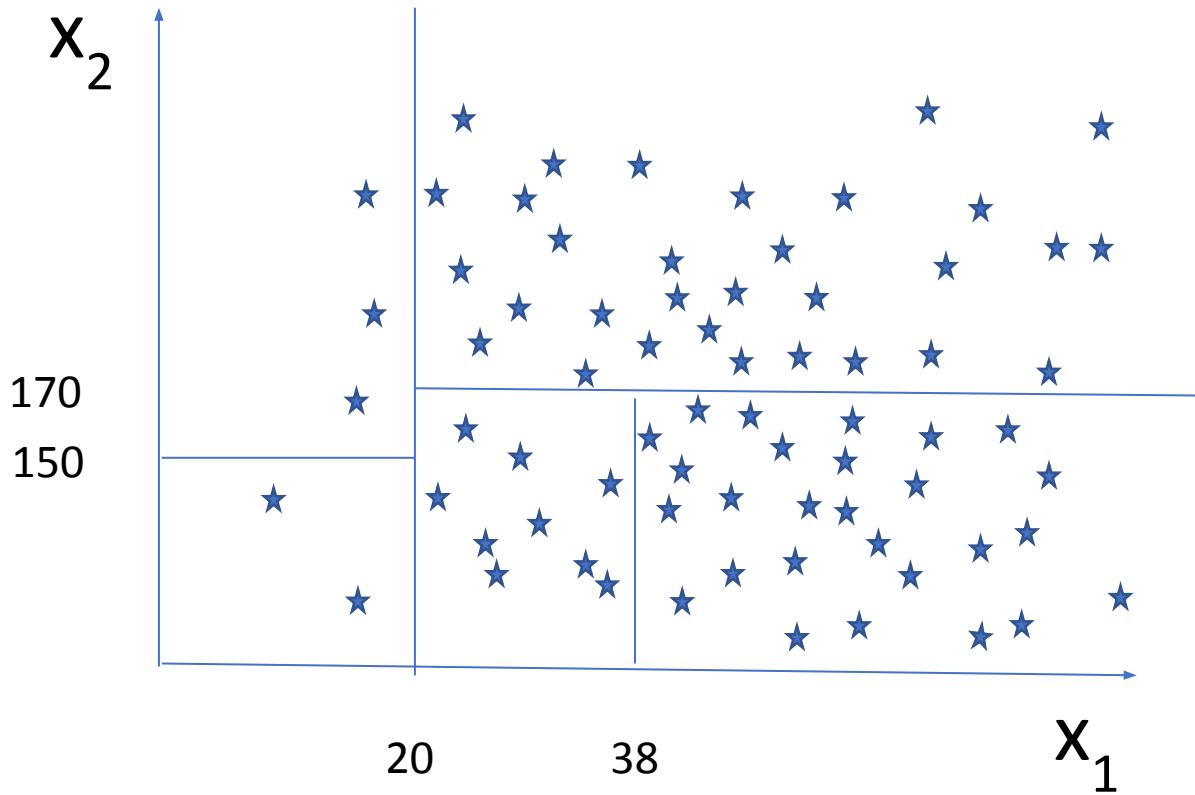
Split 2



Split 3



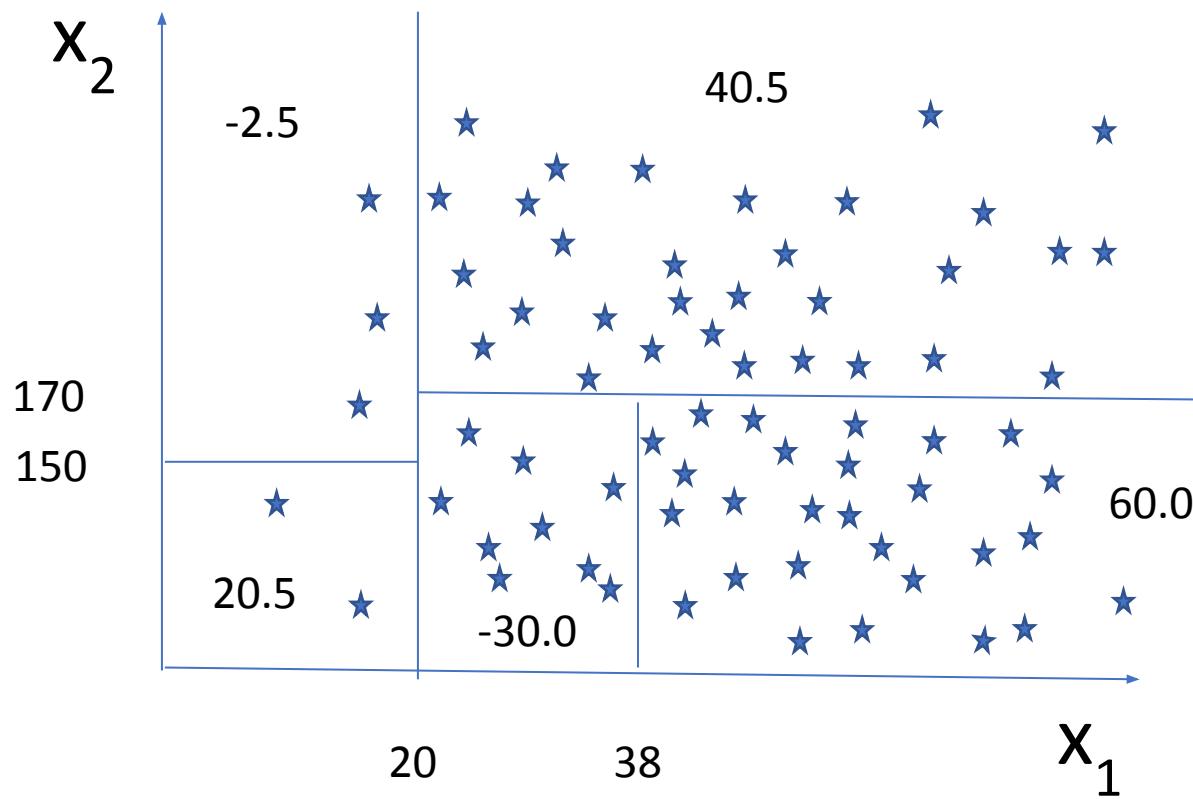
Split 4



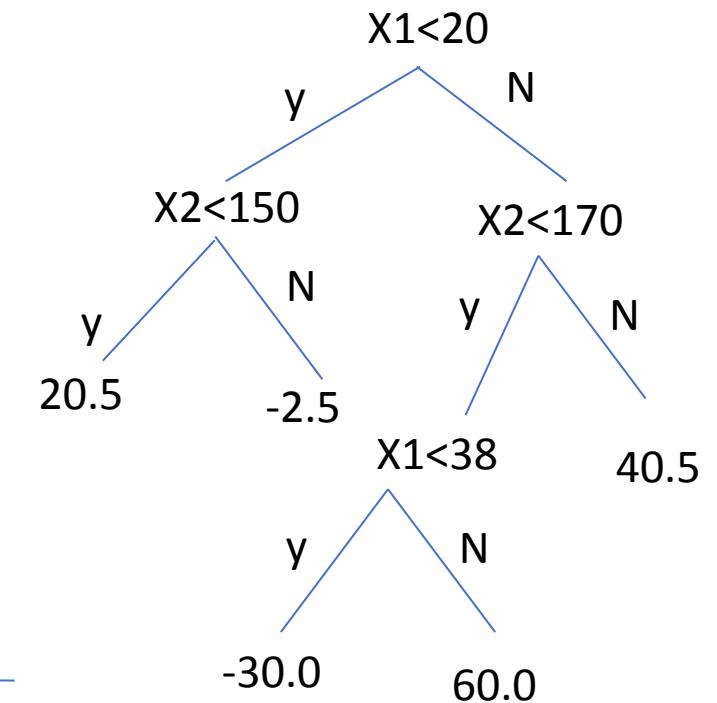
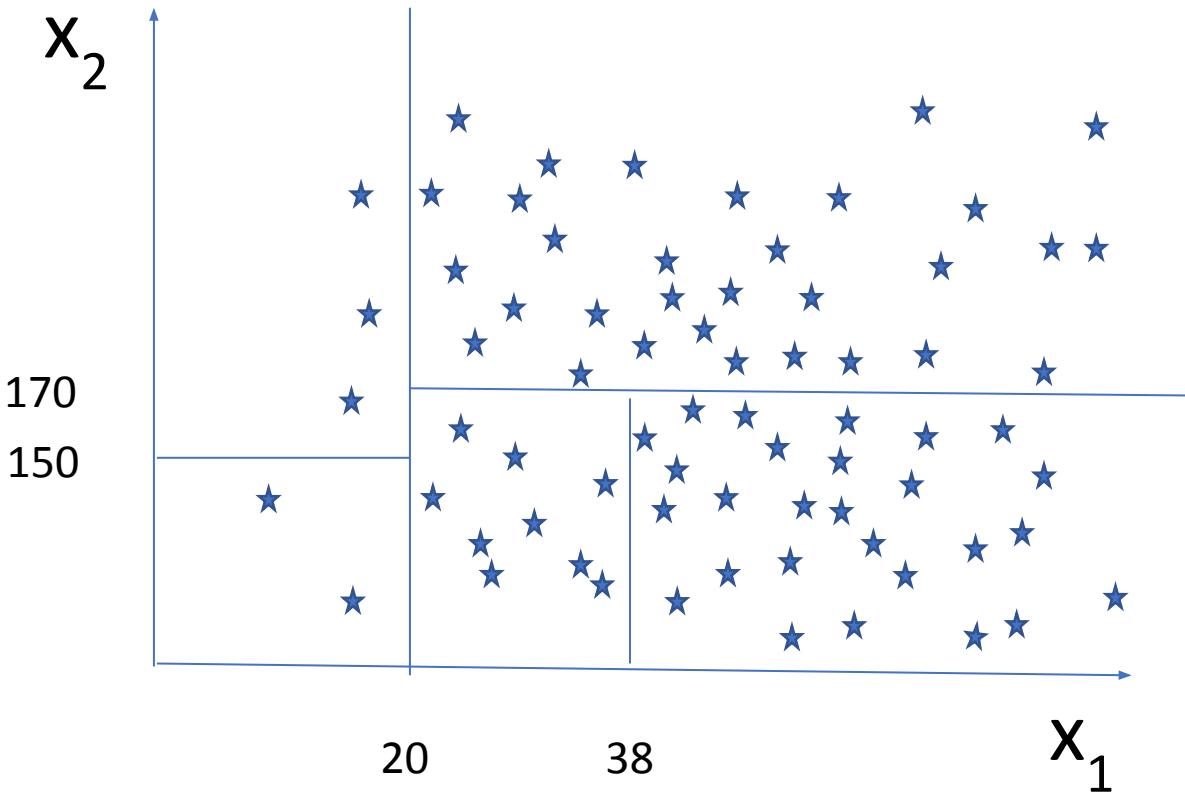
Finding Value of dependent var for a new instance

- How we get the value of a point?
- terminal leaf in which this point falls help us to determine value there
 - We take the average of y for all the points in that terminal leaf
- Default option
 - Take average of values of all the points
- Since we are now taking average of a particular segment of dataset, accuracy will be better.

Averages



Split 4



- Feature scaling is not required for decision tree and random forest model.
- Decision trees have been around for a long time and also known to suffer from bias and variance.
 - You will have a large bias with simple trees and a large variance with complex trees.

Importing the libraries

```
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Position_Salaries.csv')
```

```
x = dataset.iloc[:, 1:-1].values
```

```
y = dataset.iloc[:, -1].values
```

Position	Level	Salary
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

Other steps required

- Taking care of Missing values
- Taking care of Categorical var or labeled var
- Split dataset
- Don't need to apply feature scaling
 - It splits features in different successive ranges, and final predictions are done from different ranges of features

Training the Decision Tree Regression model on the whole dataset

```
from sklearn.tree import DecisionTreeRegressor  
regressor = DecisionTreeRegressor(random_state = 0)  
regressor.fit(x, y)
```

Predicting a new result

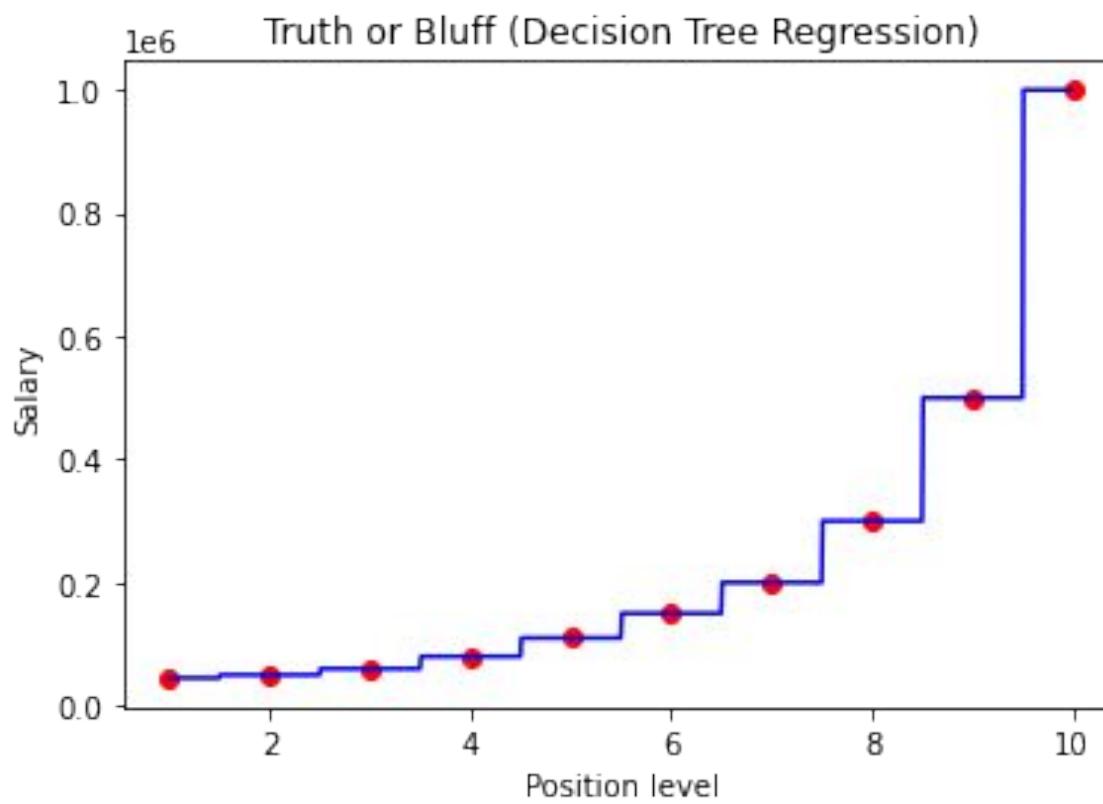
- `regressor.predict([[6.5]])`
- Output
 - `array([150000.])`
- If we have several features (say 3 features)
 - `Regressor.predict([[6.5, 30, 5]])`

Visualising the Decision Tree Regression results (higher resolution)

```
x_grid = np.arange(min(x), max(x), 0.01)
x_grid = x_grid.reshape((len(x_grid), 1))
plt.scatter(x, y, color = 'red')
plt.plot(x_grid, regressor.predict(x_grid), color = 'blue')
plt.title('Truth or Bluff (Decision Tree Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

- If dataset is multidimensional then there is no meaning of plotting it in two dimension

Output



What is done here

- Take the real results (salaries) for each position level
- For all the position levels, from position level -4.5 to position level +4.5 it predicted the salary to be the same as the position level in the middle.
- As we know different ranges of the features where the prediction is the same
 - So here all the predicted salaries in the range position level -4.5 to position level +4.5 are just the same.
- Here, Regression curve is not continuous, we jump from position level to the next one every step
- Regression trees generally have great performance for higher dimensional datasets

Choosing the right
classification model for a
particular situation

- It involves several steps and considerations.
- Here are some key factors to consider:
 - Problem type
 - Data characteristics
 - Model complexity
 - Performance metrics
 - Domain knowledge

Problem type

- The first step is to determine what type of problem you are trying to solve. Is it a binary classification problem, where you are trying to predict one of two possible outcomes? Or is it a multiclass classification problem, where you are trying to predict among several possible outcomes? Understanding the problem type is crucial in selecting an appropriate model.

popular classification models that can be used for binary classification

- Logistic Regression - A simple and widely-used model that works well when the classes are linearly separable.
- Decision Trees - A tree-based model that is easy to interpret and can handle both categorical and numerical features.
- Support Vector Machines (SVMs) - A powerful model that works well with both linear and non-linearly separable classes.
- Random Forest - An ensemble of decision trees that can handle large datasets with high dimensionality and can avoid overfitting.
- Gradient Boosting - A boosting algorithm that combines weak classifiers to form a strong classifier, it can be used to optimize a variety of loss functions and can handle noisy data.
- Neural Networks - A highly flexible and powerful model that can learn complex relationships between features and classes, it can be trained using backpropagation algorithm and various optimization techniques
- The choice of model will depend on factors such as the size and complexity of the dataset, the balance of the classes, the interpretability of the model, and the computational resources available. It's important to evaluate multiple models and choose the one that performs best on the given task.

popular classification models that can be used when dealing with multiclass classification problems

- Logistic Regression - An extension of binary logistic regression that can handle multiple classes, it can be used for small to medium-sized datasets with a moderate number of classes
- Decision Trees - A tree-based model that can handle both categorical and numerical features and can be used for both small and large datasets with a moderate number of classes
- Random Forest - An ensemble of decision trees that can handle large datasets with high dimensionality and can avoid overfitting, it can be used for a moderate number of classes
- Support Vector Machines (SVMs) - A powerful model that can handle both linear and non-linearly separable classes, it can be used for small to medium-sized datasets with a moderate number of classes
- Gradient Boosting - A boosting algorithm that combines weak classifiers to form a strong classifier, it can be used to optimize a variety of loss functions and can handle noisy data, it can be used for a moderate number of classes
- Neural Networks - A highly flexible and powerful model that can learn complex relationships between features and classes, it can be used for a large number of classes.
- The choice of model will depend on factors such as the size and complexity of the dataset, the balance of the classes, the interpretability of the model, and the computational resources available. It's important to evaluate multiple models and choose the one that performs best on the given task. Additionally, techniques such as one-vs-all, one-vs-one, or multiclass cross-entropy loss can be used to handle multiclass classification problems.

Data characteristics

- The characteristics of your data can also influence your choice of model. For example, if your data is highly imbalanced (i.e., one class has many more examples than the other), you may need to use a specialized model that can handle this type of data. Similarly, if your data has many features, you may need to consider models that can handle high-dimensional data.

In case of highly imbalanced dataset which classification model we should use

- When dealing with a highly imbalanced dataset, where one class has significantly fewer samples than the other class(es), the choice of classification model becomes crucial. Here are some models that can be used for imbalanced datasets:
- Random Forest - Random Forest is a popular ensemble model that can handle imbalanced datasets well. It can avoid overfitting and can produce reliable estimates of feature importance.
- Gradient Boosting - Gradient Boosting is a boosting algorithm that combines weak classifiers to form a strong classifier. It can handle imbalanced datasets well and is less prone to overfitting.
- Support Vector Machines (SVMs) - SVMs can handle imbalanced datasets by adjusting the penalty parameter C, which controls the trade-off between maximizing the margin and minimizing the classification error.
- Adaptive Boosting - Adaptive Boosting is a boosting algorithm that assigns higher weights to misclassified samples, forcing the model to focus more on the difficult samples. It can handle imbalanced datasets well and can improve the performance of weak classifiers.
- Resampling Techniques - Resampling techniques such as oversampling the minority class or undersampling the majority class can be used to balance the dataset before applying any classification model.
- It's important to evaluate the performance of the model using appropriate metrics such as precision, recall, and F1-score, rather than just accuracy, since accuracy can be misleading in the case of imbalanced datasets.

which classification models we should choose when multi-dimensional dataset

- Random Forest - Random Forest is an ensemble model that can handle high-dimensional datasets well. It can avoid overfitting and can produce reliable estimates of feature importance.
- Support Vector Machines (SVMs) - SVMs can handle high-dimensional datasets well, especially when using a kernel function that maps the input features to a higher-dimensional space. However, SVMs can be computationally expensive and may not scale well to very large datasets.
- Logistic Regression - Logistic Regression is a simple and interpretable model that can handle high-dimensional datasets well. It works well when the classes are linearly separable.
- Neural Networks - Neural Networks are highly flexible models that can learn complex relationships between input features and classes. They can handle high-dimensional datasets well, but can be computationally expensive and require a lot of data to train.
- Naive Bayes - Naive Bayes is a simple and fast model that can handle high-dimensional datasets well. It assumes that the input features are conditionally independent given the class, which can be a strong assumption but can work well in practice.
- It's important to evaluate the performance of the model using appropriate metrics such as precision, recall, and F1-score, and to apply dimensionality reduction techniques such as PCA or LDA to reduce the dimensionality of the dataset if necessary.

Model complexity

- Another consideration is the complexity of the model. More complex models, such as neural networks, may be able to capture more complex relationships in the data, but may also be more prone to overfitting. In contrast, simpler models, such as logistic regression, may be easier to interpret and less prone to overfitting, but may not capture all the nuances in the data.

Performance metrics

- The performance metrics that you use to evaluate your model can also influence your choice of model.
- For example, if you are interested in maximizing accuracy, you may choose a different model than if you are interested in maximizing precision or recall.

when we are interested in maximizing accuracy rather than precision or recall

- Maximizing accuracy is a common goal in many classification problems, but it may not always be the most appropriate metric to optimize, especially when dealing with imbalanced datasets. In cases where the class distribution is heavily imbalanced, accuracy can be misleading and can result in a model that simply predicts the majority class all the time.
- However, if the class distribution is balanced or only slightly imbalanced, maximizing accuracy can be a reasonable goal. In this case, models that have high accuracy and low variance, such as Random Forest or Gradient Boosting, can be good choices.

when we are interested in maximizing accuracy rather than precision or recall

- In some classification problems, accuracy may be the most important metric to optimize for. This is typically the case in problems where the cost of a false positive and false negative are roughly equal. For example:
- Image Classification: In an image classification problem, the goal is to correctly classify an image into one of several categories. In this case, the cost of a false positive or false negative may be equal, and accuracy may be the most important metric to optimize for.
- Sentiment Analysis: In a sentiment analysis problem, the goal is to predict the sentiment of a text as positive, negative, or neutral. In this case, the cost of misclassifying a text as positive or negative may be roughly equal, and accuracy may be the most important metric to optimize for.
- Customer Segmentation: In a customer segmentation problem, the goal is to group customers into different segments based on their characteristics. In this case, the cost of misclassifying a customer into the wrong segment may be roughly equal, and accuracy may be the most important metric to optimize for.

If the goal is to maximize accuracy, the following classification models can be considered

- Logistic Regression: It is a simple and interpretable linear model that can work well when the classes are linearly separable. Logistic Regression is often used as a baseline model and can achieve high accuracy on many datasets
- Random Forest: Random Forest is an ensemble model that combines multiple decision trees to make predictions. It can handle complex relationships between the input features and the target variable and can avoid overfitting. Random Forest can achieve high accuracy on a wide range of datasets.
- Gradient Boosting: Gradient Boosting is another ensemble model that can achieve high accuracy on a wide range of datasets. It combines multiple weak models to form a strong model and can handle non-linear relationships between the input features and the target variable.
- Support Vector Machines (SVMs): SVMs can be used for classification problems with linear or non-linear boundaries. They are effective at separating classes in high-dimensional spaces and can achieve high accuracy on many datasets.
- Neural Networks: Neural Networks are highly flexible models that can learn complex relationships between the input features and the target variable. They can achieve high accuracy on a wide range of datasets, but can be computationally expensive and require a lot of data to train.

In which type of problems we focus on precision or recall rather than accuracy

- Precision and recall are metrics that are often used in classification problems where the class distribution is imbalanced, meaning that one class is much more frequent than the other. In these problems, optimizing for accuracy can be misleading, as a model that simply predicts the majority class all the time can achieve high accuracy, but may not be useful in practice.
- In such cases, it may be more important to focus on precision or recall, depending on the specific needs of the problem. For example:
- Fraud Detection: In a fraud detection problem, the goal is to identify transactions that are fraudulent. Since the number of fraudulent transactions is typically much smaller than the number of legitimate transactions, precision is often more important than recall. This is because falsely accusing a legitimate transaction as fraudulent can have serious consequences, while missing a fraudulent transaction may be less harmful.
- Medical Diagnosis: In a medical diagnosis problem, the goal is to predict whether a patient has a certain condition. Since the consequences of a false positive or false negative can be serious, both precision and recall are important. However, depending on the specific condition, precision may be more important than recall, or vice versa.
- Spam Filtering: In a spam filtering problem, the goal is to identify emails that are spam. Since the number of spam emails is typically much smaller than the number of legitimate emails, recall is often more important than precision. This is because missing a spam email can result in the user being exposed to unwanted content or phishing attacks.

If the goal is to maximize precision and recall, the following classification models can be considered:

- Decision Trees: Decision Trees are simple and interpretable models that can handle non-linear relationships between the input features and the target variable. They can be used for both binary and multi-class classification problems and can be effective at maximizing precision and recall.
- Random Forest: Random Forest is an ensemble model that can handle complex relationships between the input features and the target variable. It can avoid overfitting and can produce reliable estimates of feature importance. Random Forest can be effective at maximizing precision and recall on a wide range of datasets.
- Gradient Boosting: Gradient Boosting is another ensemble model that can handle complex relationships between the input features and the target variable. It combines multiple weak models to form a strong model and can achieve high precision and recall on a wide range of datasets.
- Support Vector Machines (SVMs): SVMs can be used for classification problems with linear or non-linear boundaries. They are effective at separating classes in high-dimensional spaces and can achieve high precision and recall on many datasets.
- Naive Bayes: Naive Bayes is a simple and fast model that can work well when the input features are conditionally independent given the class. It can be effective at maximizing precision and recall on a wide range of datasets.

Domain knowledge

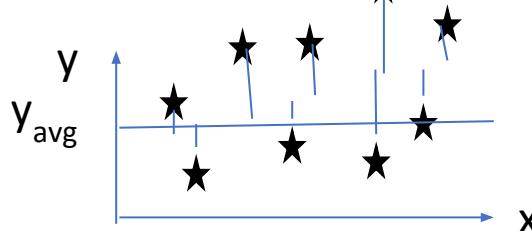
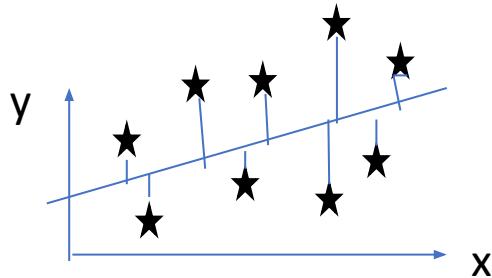
- Finally, domain knowledge can also be important in selecting a model. If you have prior knowledge about the relationships between the features and the outcome, you may be able to choose a model that is better suited to capture those relationships.

- In summary, selecting the right classification model involves considering the problem type, data characteristics, model complexity, performance metrics, and domain knowledge. It is often a iterative process of trying different models, evaluating their performance, and refining the approach based on the results.

Evaluating regression model performance & their selection

R square intuition

- Simple linear regression
 - $SS_{res} = \text{Sum } (y - y')^2 \rightarrow \min$
- Lets draw the average line instead of regression line
 - $SS_{tot} = \text{Sum}(y - y_{avg})^2 \rightarrow \min$
- $R^2 = 1 - SS_{res}/SS_{tot}$
- R^2 is normally between 0 and 1, however R^2 can be negative also
- R^2 gives how well your model is fitted to the data
 - In fact how much model is fitted than the average line that can be fitted to the data easily.



R square intuition

- $R^2 = 0$ (fitted model is equally good as average line)
- $R^2 = 1$ ($SS_{res} = 0 \rightarrow$ ideal scenario)
- $R^2 < 0$ (if your simple linear model fits the data worse than the average line)
- $R^2 \rightarrow$ goodness of fit (greater is better)

Problem in R²

- The same concept apply to multiple linear regression
 - $SS_{res} \rightarrow \min$
- Problem
 - $Y = b_0 + b_1 * x_1 + b_2 * x_2$
 - You want to check the impact of a new independent var x_3 on the model (model gets better or not (accuracy improves or not))
 - $Y = b_0 + b_1 * x_1 + b_2 * x_2 + b_3 * x_3$
- R^2 will never decrease
 - If x_3 will help in minimizing SS_{res} -> regression process will find a way to give a coefficient b_3 that will help to minimize SS_{res} then R^2 will increase
 - If x_3 has not correlation with dependent var, regression process will make its coefficient 0, then R^2 will not change
 - (rarely happens - There can always be at least a slight random correlation between independent var and dependent var thus R^2 will insignificantly increase in this case also).

Adjusted R²

- So after adding vars we don't know if those vars are helping your model or not as R² will always increase regardless of improvement or not
- Thus we have a different parameter to measure how well the model is fitted
 - $\text{Adj R}^2 = 1 - \frac{(1-R^2)}{n-p-1}$ where p-> number of regressors
 - If p increases-> denominator decreases, ratio increases, Adj R² will be decreases
 - When R² increases-> (1-R²) decreases, Adj R² will increases
- If the var is not helping the model, R² will insignificantly increase and penalization factor will actually drive Adj R² down
- If var is helping the model a lot, increase in R² will be substantial and it will overwhelm this penalization factor-> Adj R² will increase

Evaluating the Model Performance

- `from sklearn.metrics import r2_score`
- `r2_score(y_test, y_pred)`

Pros and Cons

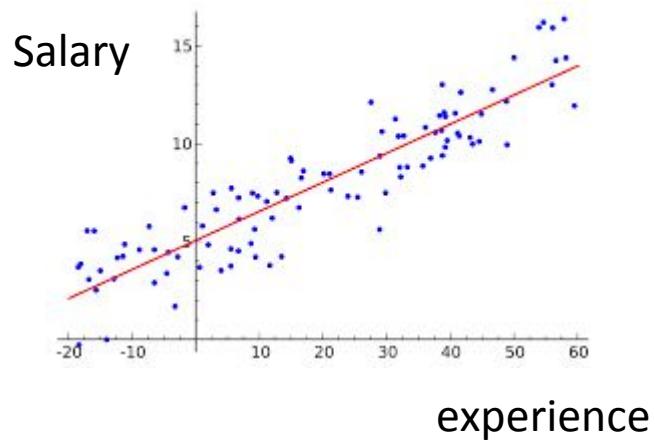
Regression Model	Pros	Cons
Linear Regression	Works on any size of dataset, gives information about relevance of features	The Linear Regression Assumptions
Polynomial Regression	Works on any size of dataset, works very well on non linear problems	Need to choose the right polynomial degree for a good bias/variance tradeoff
SVR	Easily adaptable, works very well on non linear problems, not biased by outliers	Compulsory to apply feature scaling, not well known, more difficult to understand
Decision Tree Regression	Interpretability, no need for feature scaling, works on both linear / nonlinear problems	Poor results on too small datasets, overfitting can easily occur
Random Forest Regression	Powerful and accurate, good performance on many problems, including non linear	No interpretability, overfitting can easily occur, need to choose the number of trees

Logistic Regression

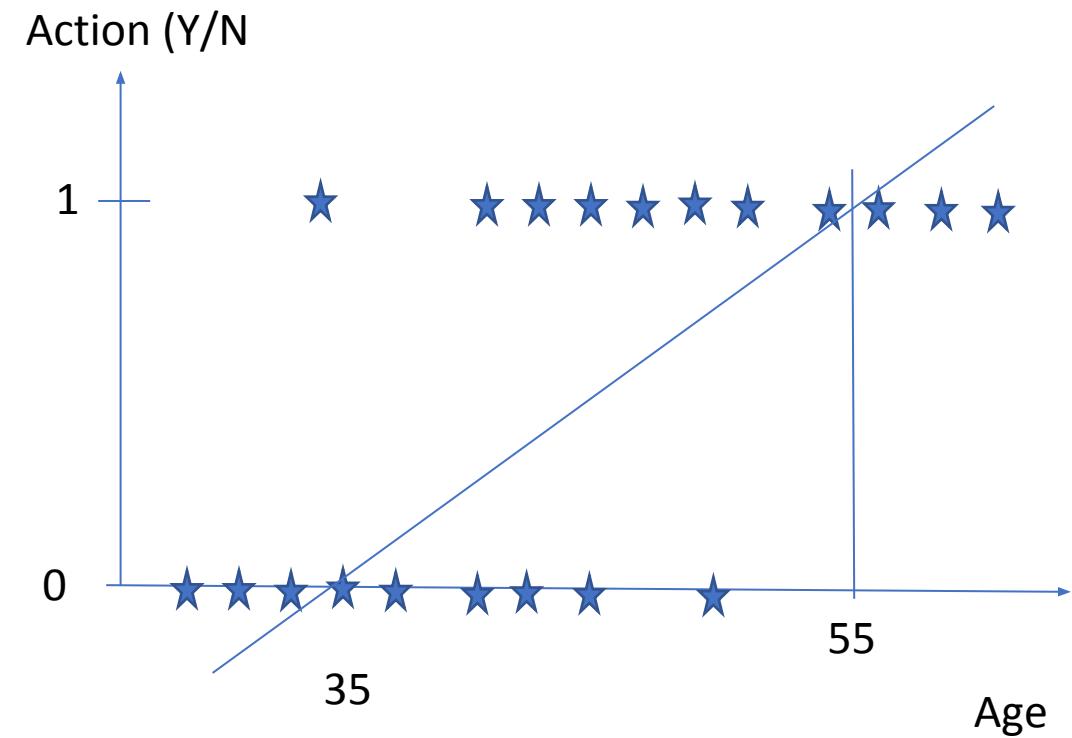
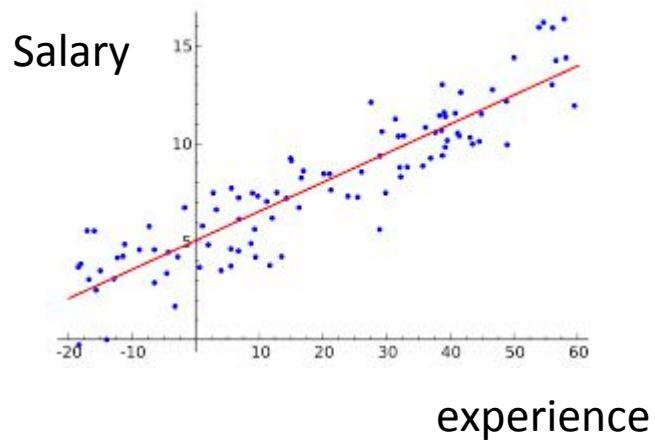
Classification

- Logistic Regression provides a 0 or 1 prediction, rather than a real value for a continuous variable.
- Unlike regression where you predict a continuous number, you use classification to predict a category
- Logistic Regression
 - is considered a linear model as it uses a linear combination of the input features
 - in fact it is generalized linear model as it applies a non-linear transformation (the sigmoid function) to the result to predict a probability.

Logistic Regression

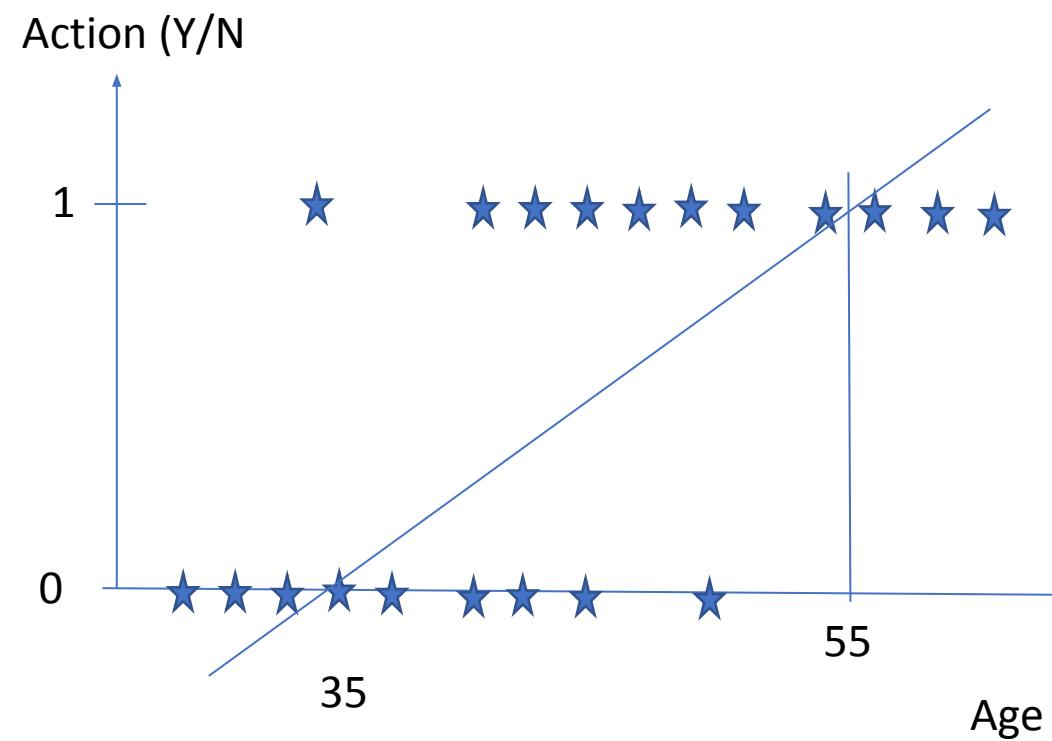


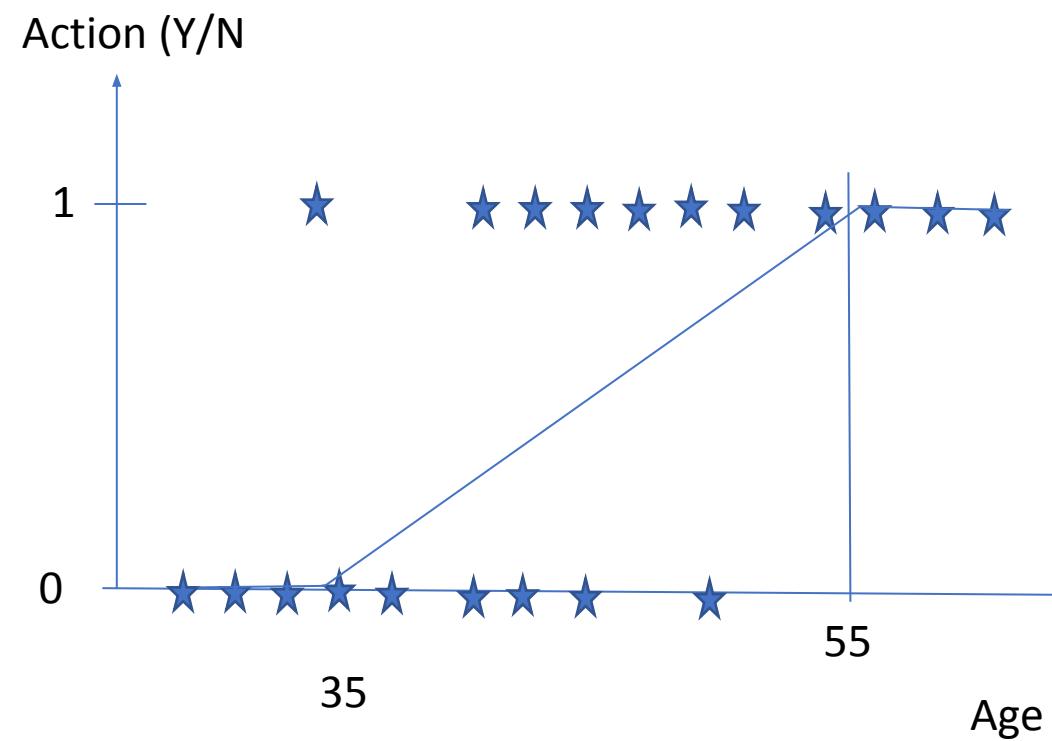
Logistic Regression



Logistic Regression

- we think of probability of taking action rather than action itself
- For known observations action is yes or no
- But for unknown observation we can't be sure
 - We can say that there is 80% chances the person will buy it
 - In that way linear regression line (middle part-between 0 and 1) makes some sense
- Persons falling between 35 and 55 there is a probability of them taking this offer and it is increasing as we move right
- But the part at the top and the bottom does not make sense at all
 - As probability can never be less than 0 or greater than 1
 - So here linear regression is trying to give us hint that people above the nominal age 55 they are very likely to take it (chances are more than 100%)
 - Anyone below 35 (left side) they will definitely not take it.





Sigmoid function

- Since interest is in predicting a Boolean answer, let us choose a function which operates within the range 0 to 1.
- Sigmoid is one such function that is extensively used for this purpose. Sigmoid function is defined as:

$$g(z) = \frac{1}{1 + e^{-z}}$$

Consider $z = \sum_{i=0}^n (\theta_i * x_i)$, and our hypothesis will be given by:

$$h_\theta(x) = g(z)$$

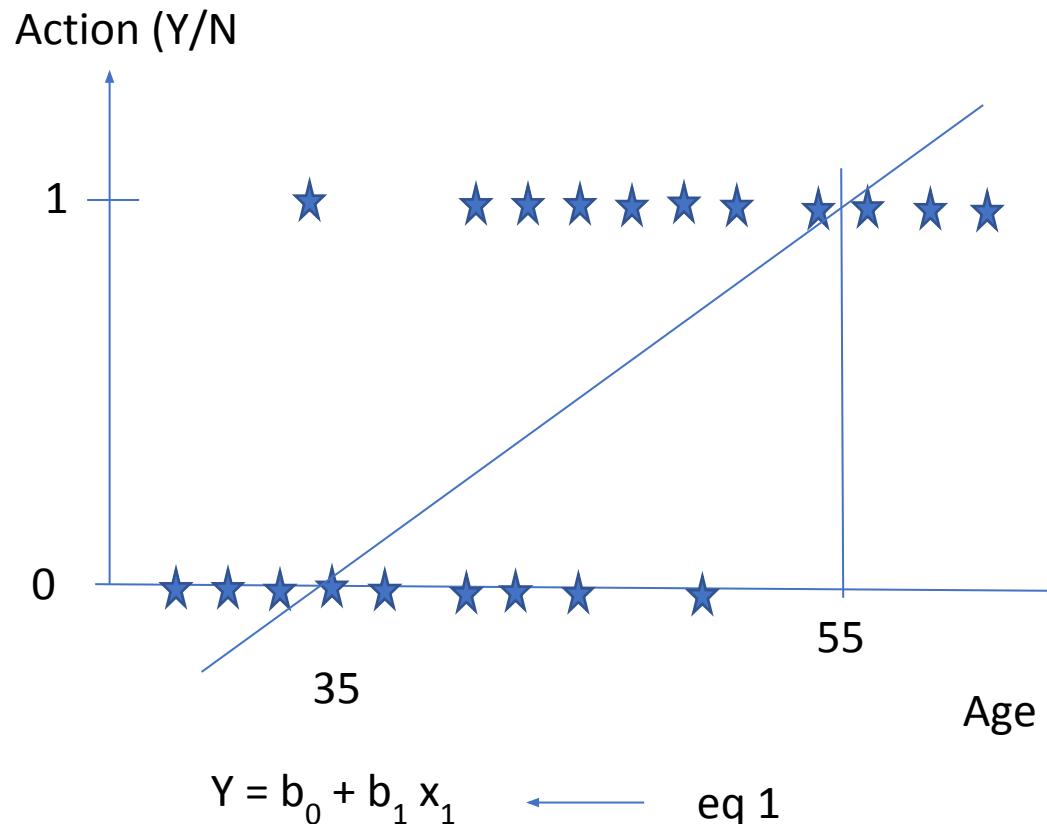
$\sum_{i=0}^n (\theta_i * x_i)$ can also be represented as $\theta^T X$

- Next Table shows the values of $g(z)$ for various values of z (or, $\theta^T X$).
- It shows that the values of $g(z)$ lie within the range 0–1.

z (or $\theta^T X$)	$g(z)$ (or, hypothesis value)
Very large negative value	~0
Below 0	Between 0 and 0.5
0	0.5
Above 0	Between 0.5 and 1
Very large positive value	~1

- This $h\theta(x)$ represents the probability that the predicted query (y) will be 1.
- If the predicted value of the hypothesis is above 0.5, you predict a value of 1, or else, predict a value of 0.

Logistic regression formula



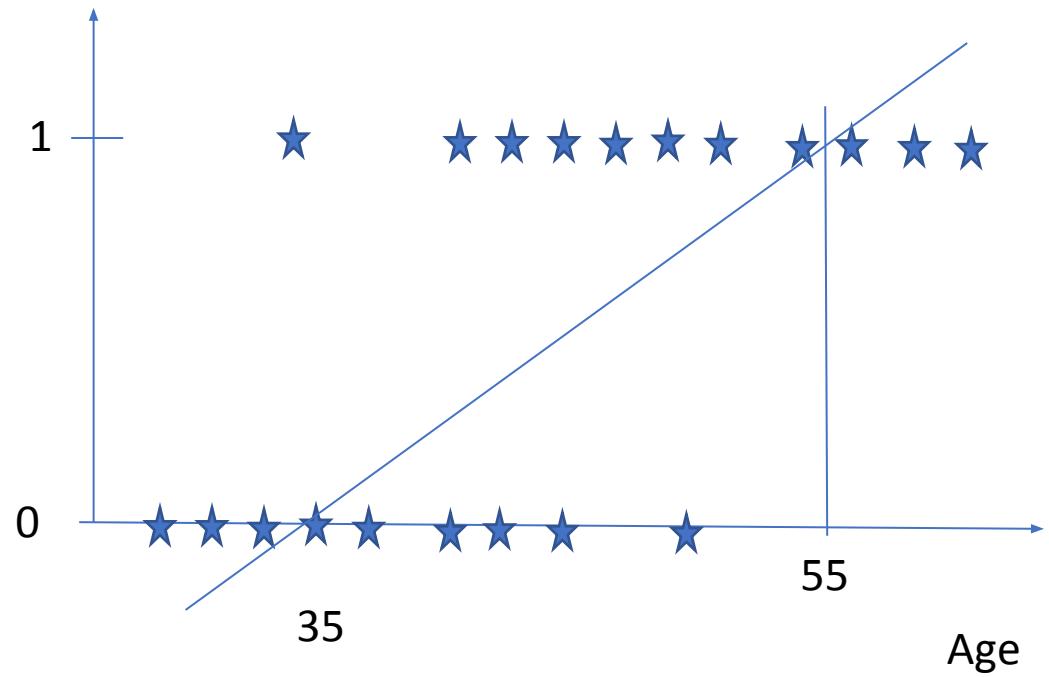
Logistic/Sigmoid function:

$$p = \frac{1}{1+e^{-y}} \quad \leftarrow \text{eq 2}$$

solve for y in eq 2 and then put this value of y in equation 1:

$$\ln \left(\frac{p}{1-p} \right) = b_0 + b_1 x$$

Action (Y/N)



Cost Function: binary cross-entropy loss function

- You could think of a cost function to apply to Logistic Regression

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h(x^i) - y^i)^2$$

- However, now that our hypothesis (h_0) is no longer linear (Sigmoid is a nonlinear function)
 - this cost function has multiple local minima and so is not very suitable.
- Let us identify a cost function that is more suitable for the current purpose.
- The cost function should be defined such that, for any given observed data,
 - the value is very high when the hypothesis does not match the observation and
 - the cost value is very low when the hypothesis matches the observation

Cost Function: binary cross-entropy loss function

- One such cost function (for a single data observation) is given by

$$\text{Cost} = -y \log(h_0(x)) - (1 - y) \log(1 - h_0(x))$$

Or, $\text{Cost} = -[y \log(h_0(x)) + (1 - y) \log(1 - h_0(x))]$

- The first term in the above equation measures the error
 - when the actual label is 1
- the second term measures the error
 - when the actual label is 0.
- Why negative sign
 - $\log(h_0(x))$ or $\log(1-h_0(x))$ will be negative as $h_0(x)$ is probability
- So the overall cost function turns out to be:

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^i \log(h(x^i)) + (1 - y^i) \log(1 - h(x^i)) \right]$$

Cost Function: binary cross-entropy loss function

- Table shows how this cost function fits our criterion of putting in
 - higher cost to situations where the hypothesis does not match the actual observation and
 - a low cost where the hypothesis matches the actual observation.

Cost corresponding to combinations of hypothesis and observed values

Y	$h(x)$	Part I $-y \log h_0(x)$	Part II $-(1 - y) \log (1 - h_0(x))$	Overall cost
$Y = 0$	$h(x) = \sim 0$	0 (because y is 0)	~ 0 (because $\log(\sim 1)$ is 0)	~ 0
	$h(x) = \sim 1$		High value (because negative of $\log \sim 0$)	High value
$Y = 1$	$h(x) = 0$	High (because negative of $\log \sim 0$)	0 (because $1 - y$ is 0)	High value
	$h(x) = 1$	~ 0 (because $\log \sim 1$)		~ 0

Gradient Descent

- So, now the main job is to find the appropriate values of θ_j , for $j = 0, 1, 2, \dots, n$.

$$\frac{\delta J}{\delta \theta_j} = \frac{1}{m} \sum_{i=1}^m (h(x^i) - y^i) \cdot x_j^i \quad (\text{for } j = 0, 1, 2, \dots, n)$$

- Note: The equation for the slope is the same for Linear Regression as well as the Logistic Regression, but the hypothesis $h(\theta)$ is not the same.
- And similar to Linear Regression, once you determine the slope of J with respect to each θ_j , update the value of θ_j using Eq.:

$$\theta_j = \theta_j - \alpha \frac{\delta J}{\delta \theta_j}$$

- and keep iterating till you converge

Multi-class Classifications

- Logistic Regression explains
 - how to classify something into one of the two classes
- How to do multi-class classification
 - Object detection is a very common example scenario of multi-class classification.

One-vs-All Classification

- The requirement to classify an object among one of the three classes can be thought of as three different classification problems:
 - Group I vs everything else (i.e., Groups II and III)
 - Group II vs everything else (i.e., Groups I and III)
 - Group III vs everything else (i.e., Groups I and II)

One-vs-All Classification

- Using three classification models
 - It will determine the decision boundary for each of these classification problems independently
 - we will have three different predictions
 - These three prediction algorithms will have the same set of features but will have different values for the parameters (θ)
 - When new data comes, each classification model will give the probability that this data corresponds to Groups I, II, and III, respectively
 - The category or group that has the highest probability wins

SoftMax function for Multiclass classification

- Recollect that for Logistic Regression, you need to first determine y , and, then, subject this y to a sigmoid operation.
- In SoftMax classification, for each class, we determine the probability of the object belonging to that class.
- One-vs-all provides a final result of the object belonging to a class
- SoftMax provides the probability of the object belonging to each class.

Basic Approach for SoftMax

- The hypothesis of softmax for multinomial classification is
 - the probability of each class is proportional to its score, normalized by the sum of the scores of all classes.
- Given a feature vector x , and a set of K classes $\{C_1, C_2, \dots, C_K\}$
 - we want to predict the class probabilities for x .
- We first compute the score for each class as a linear combination of the feature vector and a set of learnable parameters:
 - $z = Wx + b$
 - Where:
 - z is the score vector of size k , where k is the number of classes.
 - W is the weight matrix of size $n \times k$, where n is the number of input features.
 - x is the input feature vector of size n .
 - b is the bias vector of size k , which is added to the weighted sum of the input features to shift the scores.
- This linear transformation maps the input features to a set of scores that represent the probability of each class.

Example: Score calculation

- Suppose we have a binary classification problem
- The input features are represented by a vector $x = [2, 3]$,
- we want to predict the probability that the input belongs to class 1.
- After learning we got the weight matrix W of size 2×2 , where
 - each row corresponds to the weights for a class, and
 - each column corresponds to the weights for a feature.
- Let's say the weight matrix is:
- $W = [[1, 2], [3, 4]]$
- To calculate the score for class 1, we take the dot product of the input features with the first row of the weight matrix:
 - $z_1 = [2, 3] * [1, 2] = 21 + 32 = 8$
- To calculate the score for class 2, we take the dot product of the input features with the second row of the weight matrix:
 - $z_2 = [2, 3] * [3, 4] = 23 + 34 = 18$

Apply the softmax function

- To obtain a probability distribution over the classes, we apply the softmax function to the scores
 - $P_k = t = e^{Z_k}$ for each of these C classes where
- Sum up the values of P_i thus obtained over all C classes.
- Divide P_k for each class by this sum.
- The resultant number denotes the probability of the input data point labelled as the corresponding class.

	z (determined)	$t = e^z$	$t/\text{Sum}(t)$
Class I	2	7.39	0.73
Class II	1	2.72	0.27
Class III	-3	0.05	0.005
Sum		10.16	1.005

Loss function

- The loss function to be minimized during the training phase

$$\text{Loss} = \sum_{j=1}^c -y_j \log (y_{\text{predicted}})$$

- where
 - C represents number of classes being considered
 - y_j represents the actual value of class j and, hence, can only be 1 or 0
 - Note that for an input data point, only the class it belongs to contributes to the loss function, since the multiplier y is zero for all other classes.

Importing the libraries

```
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
```

```
x = dataset.iloc[:, :-1].values
```

```
y = dataset.iloc[:, -1].values
```

	Age	EstimatedSalary	Purchased
	19	19000	0
	35	20000	0
	26	43000	0
	27	57000	0
	19	76000	0
	27	58000	0
	27	84000	0
	32	150000	1
	25	33000	0

Total- 1000 entries

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split  
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)  
  
print(X_train)  
print(y_train)  
print(X_test)  
print(y_test)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
x_train = sc.fit_transform(x_train)  
x_test = sc.transform(x_test)  
  
print(X_train)  
print(X_test)
```

Training the Logistic Regression model on the Training set

```
from sklearn.linear_model import LogisticRegression  
classifier = LogisticRegression(random_state = 0)  
classifier.fit(X_train, y_train)
```

Predicting a new result

- `print(classifier.predict(sc.transform([[30,87000]])))`
- Output
 - [0]

Predicting the Test set results

```
y_pred = classifier.predict(X_test)  
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

Making the Confusion Matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score  
cm = confusion_matrix(y_test, y_pred)  
print(cm)  
accuracy_score(y_test, y_pred)
```

Visualising the Training set results

```
from matplotlib.colors import ListedColormap
x_set, y_set = sc.inverse_transform(x_train), y_train
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() -10, stop = x_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = x_set[:, 1].min() - 1000, stop = x_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(x1, x2, classifier.predict(sc.transform(np.array([x1.ravel(), x2.ravel()]).T)).reshape(x1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

Output



Visualising the Test set results

- from matplotlib.colors import ListedColormap
- X_set, y_set = sc.inverse_transform(X_test), y_test
- X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
• np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
- plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
• alpha = 0.75, cmap = ListedColormap(('red', 'green')))
- plt.xlim(X1.min(), X1.max())
- plt.ylim(X2.min(), X2.max())
- for i, j in enumerate(np.unique(y_set)):
- plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
- plt.title('Logistic Regression (Test set)')
- plt.xlabel('Age')
- plt.ylabel('Estimated Salary')
- plt.legend()
- plt.show()

Output



Multinomial Logistic Regression" or "Softmax Regression

- In multinomial logistic regression, the model parameters are learned to predict the probabilities of each class for a given set of input features. The probabilities are then normalized using the softmax function, which ensures that they sum to one, making them interpretable as probabilities.
- The cost function used in multinomial logistic regression is called the "cross-entropy loss", which penalizes the model when it predicts a low probability for the true class. The goal of the optimization is to minimize the cross-entropy loss.
- Multinomial logistic regression can handle any number of classes, making it a popular choice for multiclass classification problems. It works well for datasets with a moderate number of classes and a moderate number of features.

Regularization

Purpose of regularization

- To prevent overfitting by adding a penalty to the loss function used to optimize the model's coefficients
 - L1 regularization (Lasso)
 - L2 regularization (Ridge)
 - Elastic Net

L1 Regularization (Lasso Regularization)

- In L1 regularization, a penalty proportional to the **absolute** value of the magnitude of the coefficients is added to the loss function
- It performs both regularization and feature selection by driving some coefficients to zero

$$\text{Loss} = \sum(y - \hat{y})^2 + \lambda \sum |w_i|$$

L1 Regularization (Lasso Regularization)

Effect

- L1 regularization leads to **sparse** models, where some of the feature weights (coefficients) are **driven to zero**
 - it selects only a subset of features, performing automatic **feature selection**
- It is useful when we expect
 - some of the features to be irrelevant, or
 - the number of features is much larger than the number of data points

Use Cases

- Lasso is effective when the dataset contains irrelevant features and we want to **automatically perform feature selection**
- It works well in cases of high-dimensional data (many features)

L1 regularization does **not** help with multicollinearity

- L1 tries to **select** features.
- When two or more features are highly correlated, L1 sees them as redundant.
- Instead of sharing weight among them, it tends to pick **one** feature and push the others to **zero**.
- This behavior is unstable because small changes in the data can make L1 switch which feature it keeps.

L2 Regularization (Ridge Regularization)

- L2 regularization adds a penalty to a model for having large weight values
- A penalty proportional to the **square** of the magnitude of the coefficients is added to the loss function
- Models with extremely large coefficients tend to overfit.
- Ridge keeps the model weights small and smooth, thereby controlling the complexity of the model

$$\text{Loss} = \sum(y - \hat{y})^2 + \lambda \sum w_i^2$$

- Here:
 - w_i are the model weights
 - λ is the regularization strength
 - Larger λ forces weights to shrink more

A very small example: one feature and one weight

- we want to find weight w that minimizes
- $\text{Loss}(w) = (y - wx)^2 + \lambda w^2$

$$\frac{d}{dw} \text{Loss}(w) = 2x(y - wx) + 2\lambda w$$

$$w = \frac{xy}{x^2 + \lambda}$$

- Setting it to zero gives:
- This number is never exactly zero unless $y=0$.
The Ridge formula always produces a **shrunk but nonzero** value.

L2 Regularization (Ridge Regularization)

Effect

- L2 regularization tends to **shrink** the weights, but typically does not drive them to zero
- It smoothens the model, reducing the risk of overfitting by reducing the impact of less important features
- Works well in situations where many features contribute to the outcome

Use Cases

- Ridge regression is effective when the dataset has a high degree of multicollinearity (correlated input features)
- Helps in cases where we expect most features to contribute in some way, but we want to avoid large coefficients

L2 regularization help with multicollinearity

- When two features are correlated
 - instead of trying to pick one feature or flip between them, L2 spreads the weight across both.

Elastic Net Regularization

- Elastic Net combines **L1 (Lasso)** and **L2 (Ridge)** regularization in a single model.
- It gives the strengths of both
 - L1 helps with feature selection
 - L2 brings stability and handles correlated features well
- The loss function looks like this

$$\text{Loss} = \sum(y - \hat{y})^2 + \lambda_1 \sum |w_i| + \lambda_2 \sum w_i^2$$

- Often written using a mixing parameter α

$$\text{Loss} = \sum(y - \hat{y})^2 + \lambda \left(\alpha \sum |w_i| + (1 - \alpha) \sum w_i^2 \right)$$

- Here:
 - $\alpha = 1$ gives pure Lasso
 - $\alpha = 0$ gives pure Ridge
 - Values in between blend both penalties

Bayesian Classifiers

- Bayesian classifiers are statistical classifiers
- They can predict class membership probabilities such as the probability that a given tuple belongs to a particular class.
- Bayesian classification is based on Bayes' theorem.
- Bayesian classifiers can be classified into
 - Naive Bayes Classifier
 - Bayesian Networks

Naïve Bayes Classifier

- It is a probabilistic machine learning model
- Naïve Bayes Classifier is also called as simple Bayesian classifier
- The crux of the classifier is based on the Bayes theorem

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

How Bayes Theorem is used here

Example

Y- will buy a new product

X- features like age and salary

$$X = (x_1, x_2, x_3, \dots, x_n)$$

$$P(y|x_1, \dots, x_n) = \frac{P(x_1|y)P(x_2|y)\dots P(x_n|y)P(y)}{P(x_1)P(x_2)\dots P(x_n)}$$

- Now, we can obtain the values for each by looking at the dataset and substitute them into the equation.
- For all entries in the dataset, the denominator does not change, it remain static.
- Therefore, the denominator can be removed and a proportionality can be introduced.

$$P(y|x_1, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i|y)$$

Age	EstimatedSalary	Purchased
• 19	19000	0
• 35	20000	1
• 26	43000	0
• 32	57000	1
• 19	76000	0

Naïve Bayes Classifier

- In our case, the class variable(y) has only two outcomes, yes or no.
- There could be cases where the classification could be multivariate.
- Therefore, we need to find the class y with maximum probability

$$y = \operatorname{argmax}_y P(y) \prod_{i=1}^n P(x_i|y)$$

- Using the above function, we can obtain the class, given the predictors.

Example

- There are two machines for creating an item
- Machines work at different rate
- Items are marked with the machine number that created it so we know that particular item came from which machine
- Total items produced by both machines have few defective items also.
- What is the probability of machine 2 producing a defective item?
 - If we select any item produced by machine 2, what is the probability that it is defective?

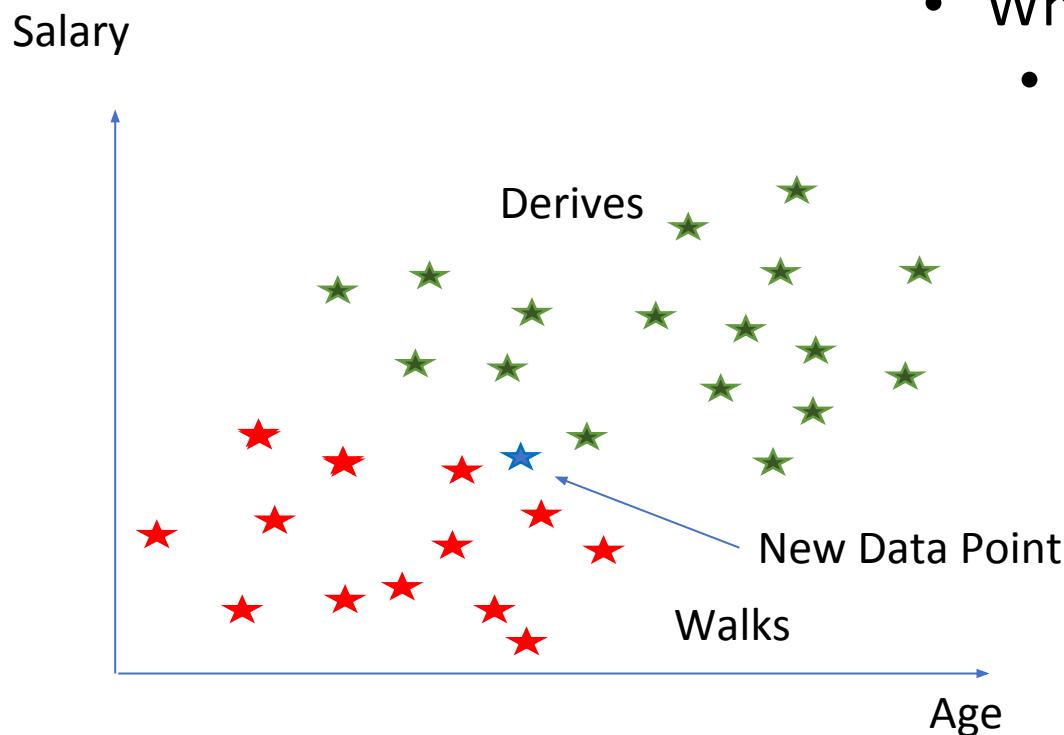
Bayes Theorem

- Machine 1: 30 items/hour
- Machine 2: 20 items/hour
- Out of all produced items:
 - 1% are defective
- Out of all defective items:
 - 50% came from machine1 and 50% came from machine2
- Question:
 - What is the probability that an item produced by machine2 is defective?
- Probability of any given item coming from machine1: $P(\text{Mach1}) = 30/50 = 0.6$
- Probability of any given item coming from machine2: $P(\text{Mach2}) = 20/50 = 0.4$
- $P(\text{Defect}) = 1\%$
- $P(\text{Mach1} | \text{Defect}) = 50\%$
- $P(\text{Mach2} | \text{Defect}) = 50\%$
- $P(\text{Defect} | \text{Mach2}) = ?$

Bayes Theorem

- $P(\text{Defect} | \text{Mach2}) = \frac{P(\text{Mach2} | \text{Defect}) * P(\text{Defect})}{P(\text{Mach2})} = \frac{.5 * 0.01}{0.4} = 0.0125 = 1.25\%$
- Frequentist interpretation
 - If machine 2 produces 1000 items then according to Bayes theorem 12 and half of them will be defective.
- $P(\text{Defect} | \text{Mach1}) = \frac{P(\text{Mach1} | \text{Defect}) * P(\text{Defect})}{P(\text{Mach1})} = \frac{.5 * 0.01}{0.6} = 0.0125 = .008\%$

Naïve Bayes Classifier



- Why Naïve?
 - Bayes Theorem has some Independence Assumptions
 - That are often times not correct therefore its kind of naïve to assume that its correct
 - e.g. Features have to be independent
 - In this example Age and salary are not independent
 - i.e. the effect of an attribute value on a given class is independent of the values of the other attributes.
 - This assumption is called classconditional independence.
 - It is made to simplify the computations involved and, in this sense, is considered “naïve.”

Step 1

- $P(\text{Walks} | X)$ where X ↳ features of new data points
 - Posterior probability of somebody being a person that walks to work given that it is placed in the position where we are placing the new observation for our dataset
 - What is the likelihood of that new observation representing a person who walks to work?

Step1

$$\bullet P(\text{Walks} | X) = \frac{P(X|\text{Walks}) * P(\text{Walks})}{P(X)}$$

Step1

$$\bullet P(\text{Walks} | X) = \frac{P(X|\text{Walks}) * P(\text{Walks})}{P(X)} \quad \xleftarrow{\text{Prior Probability}}$$

Step1

$$\bullet P(\text{Walks} | X) = \frac{P(X|\text{Walks}) * P(\text{Walks})}{P(X)} \quad \begin{matrix} \xleftarrow{\hspace{1cm}} & \text{Prior Probability} \\ \xleftarrow{\hspace{1cm}} & \text{Marginal Likelihood} \end{matrix}$$

Step1

$$\bullet P(\text{Walks} | X) = \frac{P(X|\text{Walks}) * P(\text{Walks})}{P(X)}$$

Likelihood

Prior Probability

Marginal Likelihood

Step1

$$\bullet P(\text{Walks} | X) = \frac{P(X|\text{Walks}) * P(\text{Walks})}{P(X)}$$

↓ ↓

Posterior Probability Likelihood

Prior Probability Marginal Likelihood

Step2

$$\bullet P(\text{Drives} | X) = \frac{P(X|\text{Drives}) * P(\text{Drives})}{P(X)}$$

↑ ↓

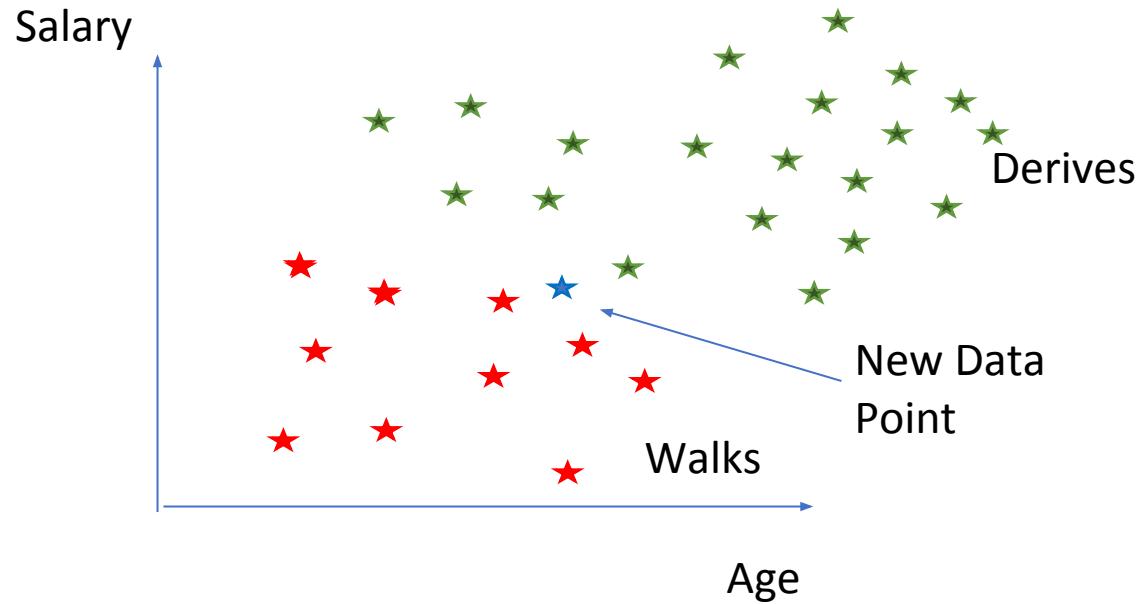
Posterior Probability Likelihood

Prior Probability Marginal Likelihood

Step3

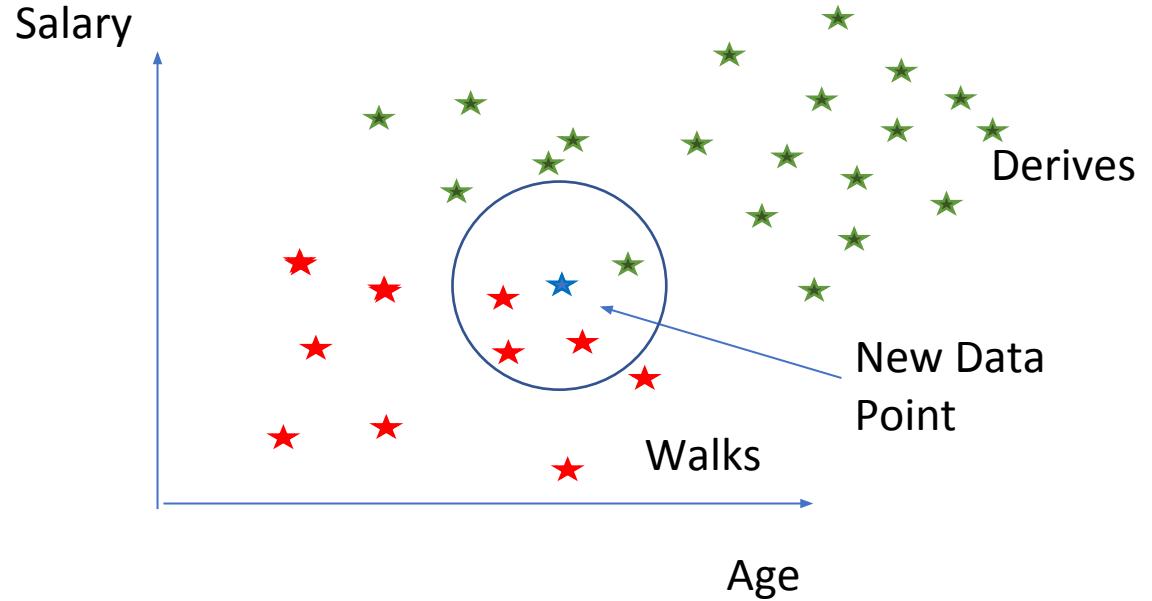
- $P(\text{Walks} | X)$ v.s. $P(\text{Drives} | X)$
- $\frac{P(X|\text{Walks}) * P(\text{Walks})}{P(X)}$ vs $\frac{P(X|\text{Drives}) * P(\text{Drives})}{P(X)}$
- $\frac{P(X|\text{Walks}) * P(\text{Walks})}{\cancel{P(X)}}$ vs $\frac{P(X|\text{Drives}) * P(\text{Drives})}{\cancel{P(X)}}$

Naïve Bayes Classifier



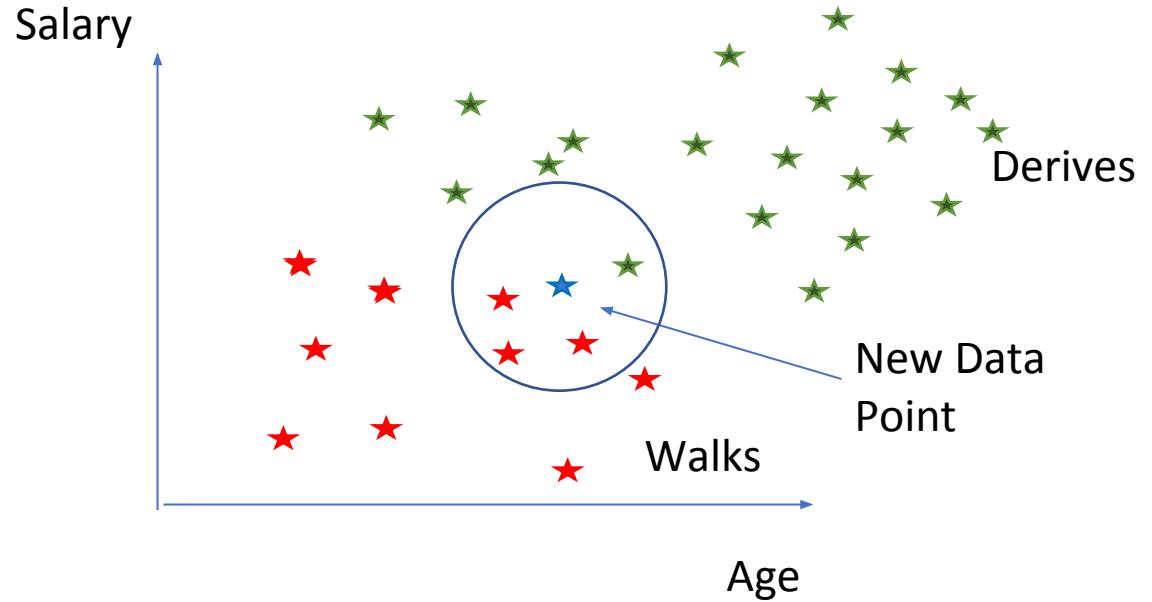
- $P(\text{Walks}) = \frac{\text{No of Walkers}}{\text{Total Observations}}$
- $P(\text{Walks}) = \frac{10}{30}$

Naïve Bayes Classifier



- $P(X)$: the likelihood that a randomly selected point from this dataset will exhibit the features similar to the datapoint that we are about to add
- Draw a circle around the data point
 - Anything in that circle is deemed to be similar to our datapoint
- $P(X)$: If we throw a random datapoint into this dataset, what is the likelihood that it will fall into the circle (it will exhibit features similar to the features of the point that we are about to add)
- To find $P(X)$ Remove the new datapoint
- $$P(X) = \frac{\text{No of Similar Observations}}{\text{Total Observations}}$$
- $$P(X) = \frac{4}{30}$$

Naïve Bayes Classifier



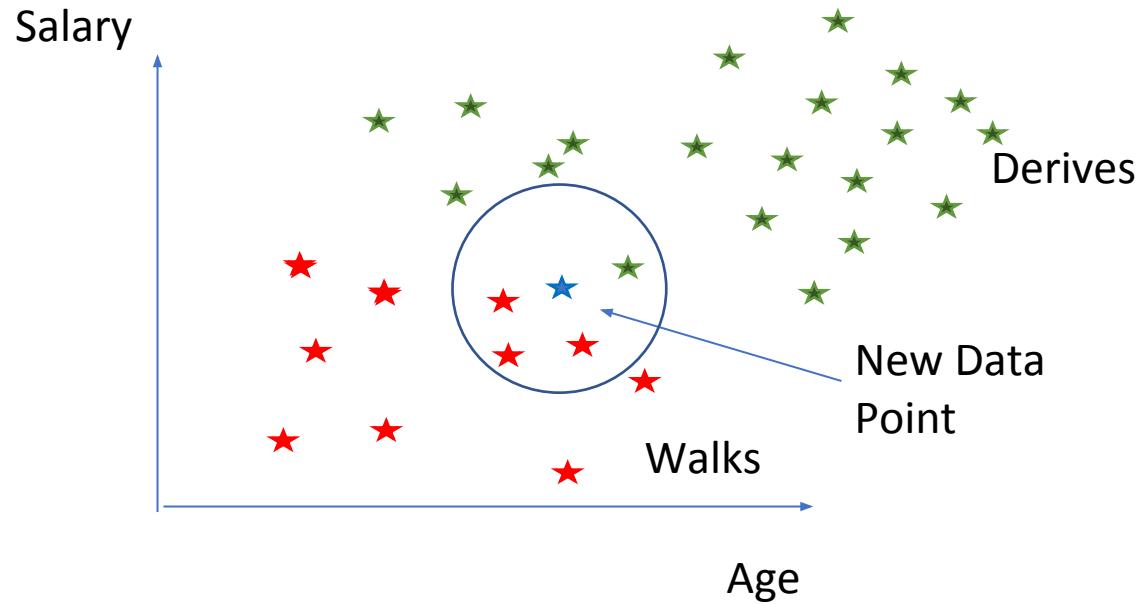
- $P(X|Walks) =$

$$\frac{\text{No of Similar Observations Among those who walks}}{\text{Total No of Walkers}}$$

- $P(X|Walks) = \frac{3}{10}$

Naïve Bayes Classifier (Step 1)

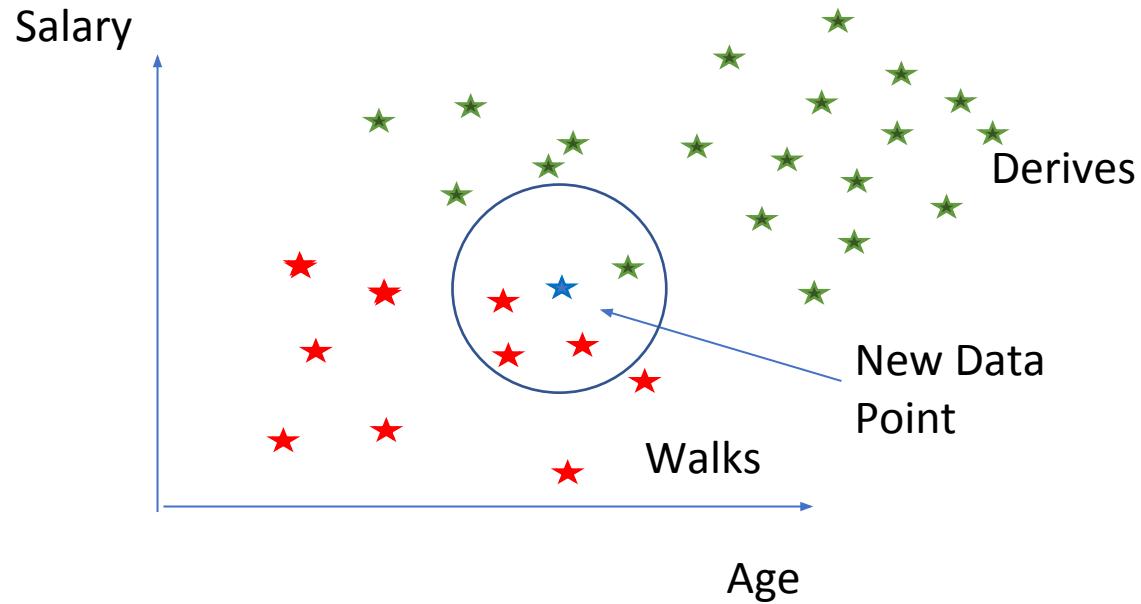
- $P(\text{Walks} | X) =$



$$\frac{\frac{3}{10} * \frac{10}{30}}{\frac{4}{30}} = 0.75$$

Naïve Bayes Classifier (Step 2)

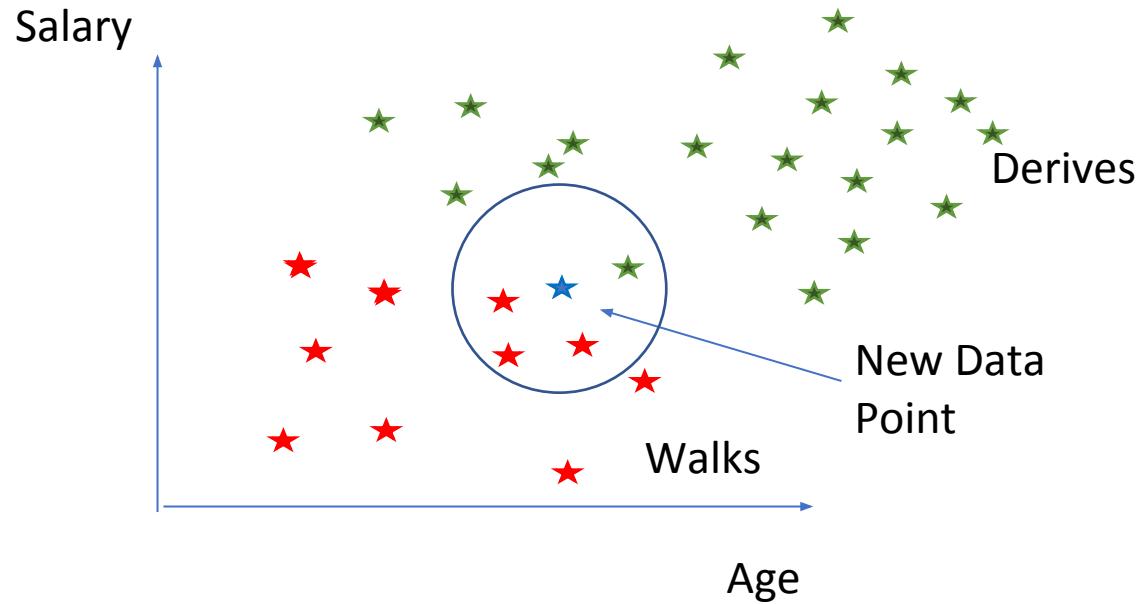
- $P(\text{Drives} | X) =$



$$\frac{\frac{1}{20} * \frac{20}{30}}{\frac{4}{30}} = 0.25$$

Naïve Bayes Classifier (Step 3)

- $P(\text{Walks} | X) > P(\text{Drives} | X)$



If more than two classes

- In case of two classes
 - $P(\text{Walks} | X)$ vs $P(\text{Drives} | X)$
 - We have calculate only one and check if its $>.50$
 - No need to calculate second one
- In case if n classes
 - We need to calculate at least $n-1$ probabilities

Bayes Theorem

- Bayes' theorem is named after Thomas Bayes
- A statistical principle for combining prior knowledge of classes with new evidence gathered from data.
- Let X be a data tuple.
 - In Bayesian terms, X is considered “evidence.”
 - As usual, it is described by measurements made on a set of n attributes.
- Let H be some hypothesis such as that the data tuple X belongs to a specified class C .
- For classification problems, we want to determine $P(H|X)$
 - the probability that the hypothesis H holds given the “evidence” or observed data tuple X .
- In other words, we are looking for the probability that tuple X belongs to class C , given that we know the attribute description of X .

Posteriori probability

- $P(H|X)$ is the posterior probability, or a posteriori probability, of H conditioned on X .
- For example, suppose our world of data tuples is confined to customers described by the attributes age and income, respectively, and that 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 $P(H|X)$ reflects the probability that customer X will buy a computer given that we know the customer's age and income.

Prior Probability

- 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.
- The posterior probability, $P(H|X)$, is based on more information (e.g., customer information) than the prior probability, $P(H)$, which is independent of X

Other probabilities

- Similarly, $P(X|H)$ is the posterior probability of X conditioned on H .
 - That is, it is the probability that a customer, X , is 35 years old and earns \$40,000, given that we know the customer will buy a computer.
- $P(X)$ is the prior probability of X .
 - Using our example, it is the probability that a person from our set of customers is 35 years old and earns \$40,000.
- How are these probabilities estimated?
 - $P(H)$, $P(X|H)$, and $P(X)$ may be estimated from the given data
- Bayes' theorem is useful in that it provides a way of calculating the posterior probability, $P(H|X)$, from $P(H)$, $P(X|H)$, and $P(X)$.

Bayes' Theorem

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

Working of Naïve Bayesian Classifier

- 1. Input: 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, $X = (x_1, x_2, \dots, x_n)$, depicting n measurements made on the tuple from n attributes, respectively, A_1, A_2, \dots, A_n
- Suppose that there are m classes, C_1, C_2, \dots, C_m .
- 2. 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
$$P(C_i|X) > P(C_j|X) \quad \text{for } 1 \leq j \leq m, j \neq i.$$
 - The class C_i for which $P(C_i|X)$ is maximized is called the *maximum posterior hypothesis*.
 - By Bayes Theorem

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

Working of Naïve Bayesian Classifier

- 3. As $P(\mathbf{X})$ is constant for all classes, only $P(\mathbf{X}|\mathbf{C}_i)P(\mathbf{C}_i)$ 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, $P(\mathbf{C}_1) = P(\mathbf{C}_2) = \dots = P(\mathbf{C}_m)$, and we would therefore maximize $P(\mathbf{X}|\mathbf{C}_i)$. Otherwise, we maximize $P(\mathbf{X}|\mathbf{C}_i)P(\mathbf{C}_i)$.
- Note that the class prior probabilities may be estimated by $P(\mathbf{C}_i) = |\mathbf{C}_{i,D}| / |\mathbf{D}|$, where $|\mathbf{C}_{i,D}|$ is the number of training tuples of class \mathbf{C}_i in \mathbf{D} .

Working of Naïve Bayesian Classifier

- 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,

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

Working of Naïve Bayesian Classifier

- How to estimate the probabilities $P(x_1 | C_i), P(x_2 | C_i), \dots, P(x_n | C_i)$
- We can easily estimate the probabilities $P(x_1 | C_i), P(x_2 | C_i), \dots, P(x_n | C_i)$ from the training tuples.
- 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:
 - i) 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 .
 - ii) If A_k is continuous-valued, then we need to do a bit more work, but the calculation is pretty straightforward.

Working of Naïve Bayesian Classifier

- A continuous-valued attribute is typically assumed to have a Gaussian distribution with a mean and standard deviation, defined by

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

- So that

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

- Where μ_{C_i} and σ_{C_i} are mean and standard deviation of the values of attribute A_k for training tuples of class C_i .

example: how Naive Bayes calculates the probability of a feature given each class label

Age	Income	Class
30	50,000	Yes
25	70,000	Yes
35	80,000	Yes
45	60,000	No
40	70,000	No

- To calculate the likelihood of a feature given a class label, Naive Bayes computes the frequency of each value of that feature for each class label.
- For example, to calculate the likelihood of the "age" feature given the "Yes" class label, we count the number of times each age appears in the "Yes" class, and divide by the total number of instances in the "Yes" class.
- We can then do the same for the "No" class.
- In this example, we can see that all three instances in the "Yes" class have a different age value, so the frequency for each age is $1/3$.
- In the "No" class, there are two instances with age values of 40 and 45, respectively, so the frequency for each of these values is $1/2$.

Age	Yes Count	Yes Freq	No Count	No Freq
25	1	1/3	0	0
30	1	1/3	0	0
35	1	1/3	0	0
40	0	0	1	1/2
45	0	0	1	1/2

- We can then do the same calculation for the "income" feature given each class label, and obtain the likelihood probabilities for each feature and class label.
- These likelihood probabilities can then be combined with the prior probabilities of each class to compute the posterior probabilities and make predictions for new instances.

Working of Naïve Bayesian Classifier

- Example:
- let $X = (35, \$40,000)$, where A_1 and A_2 are the attributes *age* and *income*, respectively.
- Let the class label attribute be *buys computer*
- The associated class label for X is *yes* (i.e., *buys computer* = *yes*)
- Let's suppose that *age* has not been discretized and therefore exists as a continuous-valued attribute.
- Suppose that from the training set, we find that customers in D who buy a computer are 38 ± 12 years of age.
 - In other words, for attribute *age* and this class, we have $\mu = 38$ years and $\sigma = 12$.
- We can plug these quantities, along with $x_1 = 35$ for our tuple X , into previous Eq. to estimate $P(\text{age} = 35 | \text{buys computer} = \text{yes})$.

Working of Naïve Bayesian Classifier

- 5. To predict the class label of X , $P(X|C_i)P(C_i)$ is evaluated for each class C_i .
- The classifier predicts that the class label of tuple X is the class C_i if and only if
 - $P(X|C_i)P(C_i) > P(X|C_j)P(C_j)$ for $1 \leq j \leq m, j \neq i$.

Predicting a class label using naïve Bayesian classification: 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

Predicting a class label using na“ive Bayesian classification: Example

- 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 $X = (\text{age} = \text{youth}, \text{income} = \text{medium}, \text{student} = \text{yes}, \text{credit rating} = \text{fair})$
- We need to maximize $P(X|C_i)P(C_i)$, for $i = 1, 2$.
- $P(C_i)$, the prior probability of each class, can be computed based on the training tuples:
 - $P(\text{buys computer} = \text{yes}) = 9/14 = 0.643$
 - $P(\text{buys computer} = \text{no}) = 5/14 = 0.357$

Predicting a class label using naïve Bayesian classification: Example

- To compute $P(X|Ci)$, for $i = 1, 2$, we compute the following conditional probabilities:
- $P(\text{age} = \text{youth} | \text{buys computer} = \text{yes}) = 2/9 = 0.222$
- $P(\text{age} = \text{youth} | \text{buys computer} = \text{no}) = 3/5 = 0.600$
- $P(\text{income} = \text{medium} | \text{buys computer} = \text{yes}) = 4/9 = 0.444$
- $P(\text{income} = \text{medium} | \text{buys computer} = \text{no}) = 2/5 = 0.400$
- $P(\text{student} = \text{yes} | \text{buys computer} = \text{yes}) = 6/9 = 0.667$
- $P(\text{student} = \text{yes} | \text{buys computer} = \text{no}) = 1/5 = 0.200$
- $P(\text{credit rating} = \text{fair} | \text{buys computer} = \text{yes}) = 6/9 = 0.667$
- $P(\text{credit rating} = \text{fair} | \text{buys computer} = \text{no}) = 2/5 = 0.400$

- Using these probabilities, we obtain
- $P(X|buys\ computer = yes) = P(age = youth | buys\ computer = yes)$
 $\quad \times P(income = medium | buys\ computer = yes)$
 $\quad \times P(student = yes | buys\ computer = yes)$
 $\quad \times P(credit\ rating = fair | buys\ computer = yes)$
 $\quad = 0.222 \times 0.444 \times 0.667 \times 0.667 = 0.044.$
- Similarly, $P(X|buys\ computer = no) = 0.600 \times 0.400 \times 0.200 \times 0.400 = 0.019$
- To find the class, C_i , that maximizes $P(X|C_i)P(C_i)$, we compute
 - $P(X|buys\ computer = yes)P(buys\ computer = yes) = 0.044 \times 0.643 = 0.028$
 - $P(X|buys\ computer = no)P(buys\ computer = no) = 0.019 \times 0.357 = 0.007$
- Therefore, the naïve Bayesian classifier predicts $buys\ computer = yes$ for tuple X .

What if I encounter probability values of zero?"

- what happens if we should end up with a probability value of zero for some $P(x_k | C_i)$?
- For example if there are no training tuples representing students for the class *buys computer = no*, resulting in $P(\text{student} = \text{yes} | \text{buys computer} = \text{no}) = 0$?
- Plugging this zero value into previous Eq. would return a zero probability for $P(X | C_i)$, even though, without the zero probability, we may have ended up with a high probability, suggesting that X belonged to class C_i !
- A zero probability cancels the effects of all the other (posteriori) probabilities (on C_i) involved in the product.
- If we have, say, q counts to which we each add one, then we must remember to add q to the corresponding denominator used in the probability calculation.

What if I encounter probability values of zero?"

- simple trick to avoid this problem
- We can assume that our training database,
- D , is so large that adding one to each count that we need would only make a negligible difference in the estimated probability value, yet would conveniently avoid the case of probability values of zero.
- This technique for probability estimation is known as the **Laplacian correction** or **Laplace estimator**, named after Pierre Laplace, a French mathematician who lived from 1749 to 1827.

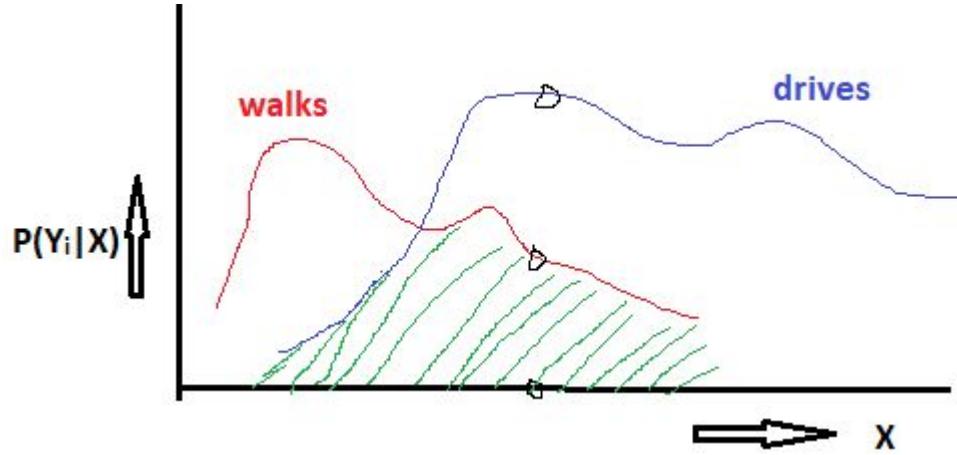
What if I encounter probability values of zero?" Example: Using the Laplacian correction to avoid computing probability values of zero

- Suppose that for the class *buys computer = yes* in some training database, D , containing 1000 tuples, we have 0 tuples with *income = low*, 990 tuples with *income = medium*, and 10 tuples with *income = high*.
- The probabilities of these events, without the Laplacian correction, are 0, 0.990 (from 990/1000), and 0.010 (from 10/1000), respectively.
- Using the Laplacian correction for the three quantities, we pretend that we have 1 more tuple for each income-value pair.
- In this way, we instead obtain the following probabilities (rounded up to three decimal places):

$$1/1003 = 0.001, 991/1003 = 0.988, \text{ and } 11/1003 = 0.011, \text{ respectively.}$$

- The “corrected” probability estimates are close to their “uncorrected” counterparts, yet the zero probability value is avoided.

Bayes Minimum Error Classification



- If($p(\text{walks} | X) > p(\text{drives} | X)$ $\Rightarrow X \in \text{walks}$)
- So we will decide $X \in \text{walks}$
- But there is a finite probability that $X \in \text{drives}$
 - This is the probability of error.
- i.e.
 - $P(\text{error} | X) = \min\{p(\text{walks} | X), p(\text{drives} | X)\}$
- As we are deciding based on minimizing the error it is known as
- Total error of classification= the area under this curve= integration of $\min\{p(\text{walks} | X), p(\text{drives} | X)\}$
 x from $-\infty$ to $+\infty$

Bayes Minimum Risk Classifier

- More general than bayes minimum error classifier
- ω_i : $i=1 \dots C$ (classes)
- a_j : $j=1 \dots K$ (actions/predictions)
- X - d-dimensional feature vector
- Why is it more general
 - Because of introduction of loss function $\lambda(a_i|\omega_j)$
 - If we take an action a_i where the true class is ω_j then the Loss that we incur is $\lambda(a_i|\omega_j)$

Bayes Minimum Risk Classifier

- Here, It is assumed that for every prediction, there is risk involved in it
 - So we compute the value of risk for every decision, and we take the decision that gives you the minimum risk
 - Given X , if we take action α_i , so the risk involved in taking action α_i given X
 - $R(\alpha_i|X) = \sum_{\forall j} \lambda(\alpha_i|\omega_j) \cdot P(\omega_j|X)$ if the true state of nature is ω_j
 - So we will calculate $R(\alpha_i|X)$ for all $i=1\dots K$ and take that action for which this risk is minimum.

Relationship between BMRC and BMEC

- Let us assume we have the 0,1 loss function
 - $\lambda(\alpha_i | \omega_j) = \lambda_{ij}$ (let us represent like this)
 - $\lambda_{ij} = 0$ if $i=j$ (i.e. we are taking correct decision, it belongs to ω_j and we are saying that it belongs to ω_j)
= 1 if $i \neq j$
 - Risk involved in taking action α_i given X ,
 - $R(\alpha_i | X) = \sum_{\forall j} \lambda_{ij} \cdot P(\omega_j | X) = \sum_{j \neq i} P(\omega_j | X)$ as for $i=j \lambda_{ij} = 0$
= $1 - P(\omega_i | X)$
 - In case of BMEC we maximize $P(\omega_i | X)$ while in case of BMRC we minimize $R(\alpha_i | X)$ that is same here, so
 - in this case (0,1 loss function) both BMRC and BMEC are same.
- In general for taking different types of wrong decisions, the loss function will not be same.

Discriminant function

- Or in other words we define a function $g_i(X)$ for class ω_i
- $X \rightarrow$ unknown feature vector
- For each class i , we compute $g_i(X)$ and for whichever class I , this function $g_i(X)$ is maximum, we will classify X to that corresponding class.
- So we want to design this function $g_i(X)$ for all ω_i
- What are the possible options
 - $g_i(X) = P(\omega_i|X) = P(X|\omega_i) \cdot P(\omega_i)$ in case of BMEC
 - $g_i(X) = -R(\alpha_i|X)$ in case of BMRC

Applications and pros and cons

- Naive Bayes algorithms are mostly used in
 - sentiment analysis, spam filtering, recommendation systems etc.
- They are fast and easy to implement
- But their biggest disadvantage is that the requirement of predictors to be independent
 - In most of the real life cases, the predictors are dependent, this hinders the performance of the classifier

- Naive Bayes is an eager algorithm that learns a probabilistic model from the training data to make predictions.
- During training phase it learns and calculate
 - the prior probability of each class label, which is the frequency of each class label in the training data.
 - Then, for each feature, Naive Bayes calculates the probability of that feature given each class label, which is known as the likelihood.

Bayesian Network

- Bayesian Network (BN) is a graphical model that represents the probabilistic relationships among a set of variables using a Directed Acyclic Graph (DAG).
- Each node represents
 - a random variable,
- and each directed edge represents
 - a probabilistic dependency between variables.
- It encodes conditional dependencies
- The joint probability distribution (JPD) over all variables can be factored as a product of local conditional probabilities:

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^n P(X_i | \text{Parents}(X_i))$$

Bayesian Network Classifier

- A Bayesian Network Classifier is a Bayesian Network in which one variable is designated as the class variable, and others are features (or attributes).
- Instead of assuming independence (like Naïve Bayes),
- we allow dependencies among features
 - as long as they are captured by the DAG.
- Example
 - Weather and Playing Tennis” Classifier
 - Let’s predict whether a person will play tennis (Yes/No) based on Weather, Temperature, and Humidity.

Step 1: Variables

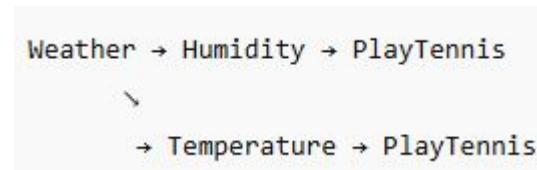
- C: PlayTennis (Yes/No) — *class variable*
- W: Weather (Sunny, Rainy)
- H: Humidity (High, Normal)
- T: Temperature (Hot, Mild, Cool)

Step 2: Define Dependencies

Suppose from experience we know:

- The **weather** affects **humidity**.
- The **temperature** depends on **weather**.
- The decision to **play tennis** depends on **humidity** and **temperature**.

We can represent this as a DAG:



Step 3: Joint Probability Factorization

Based on the DAG, the joint probability is:

$$P(W, H, T, C) = P(W) P(H | W) P(T | W) P(C | H, T)$$

Step 4: How Classification Works

When we get new data (e.g., Weather = Sunny, Humidity = High, Temperature = Hot), we compute the posterior probability of each class (Play = Yes or No):
 $P(C=\text{Yes} | W, H, T) \propto P(W) P(H | W) P(T | W) P(C=\text{Yes} | H, T)$ and similarly for C=No
Then choose the class with the higher posterior probability.

Step 5: Sample Numbers (for illustration)

$$P(W=\text{Sunny})=0.6,$$

$$P(H=\text{High} \mid W=\text{Sunny})=0.7,$$

$$P(T=\text{Hot} \mid W=\text{Sunny})=0.8,$$

$$P(C=\text{Yes} \mid H=\text{High}, T=\text{Hot})=0.1,$$

$$P(W=\text{Rainy})=0.4$$

$$P(H=\text{High} \mid W=\text{Rainy})=0.2$$

$$P(T=\text{Hot} \mid W=\text{Rainy})=0.3$$

$$P(C=\text{No} \mid H=\text{High}, T=\text{Hot})=0.9$$

If Weather = Sunny, Humidity = High, Temperature = Hot, then:

$$P(C=\text{Yes}) \propto (0.6)(0.7)(0.8)(0.1)=0.0336$$

$$P(C=\text{No}) \propto (0.6)(0.7)(0.8)(0.9)=0.3024$$

So the model predicts **PlayTennis = No**

Importing the libraries

- `import numpy as np`
- `import matplotlib.pyplot as plt`
- `import pandas as pd`

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
```

```
x = dataset.iloc[:, :-1].values
```

```
y = dataset.iloc[:, -1].values
```

	Age	EstimatedSalary	Purchased
	19	19000	0
	35	20000	0
	26	43000	0
	27	57000	0
	19	76000	0
	27	58000	0
	27	84000	0
	32	150000	1
	25	33000	0

Total- 1000 entries

Splitting the dataset into the Training set and Test set

- `from sklearn.model_selection import train_test_split`
- `x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)`

Feature Scaling

- `from sklearn.preprocessing import StandardScaler`
- `r`
- `sc = StandardScaler()`
- `X_train = sc.fit_transform(X_train)`
- `X_test = sc.transform(X_test)`

Training the Naive Bayes model on the Training set

- `from sklearn.naive_bayes import GaussianNB`
- `classifier = GaussianNB()`
- `classifier.fit(X_train, y_train)`

Predicting the Test set results

- `y_pred = classifier.predict(X_test)`
- `print(np.concatenate((y_pred.reshape(len(y_pred)), 1), y_test.reshape(len(y_test), 1)), 1))`

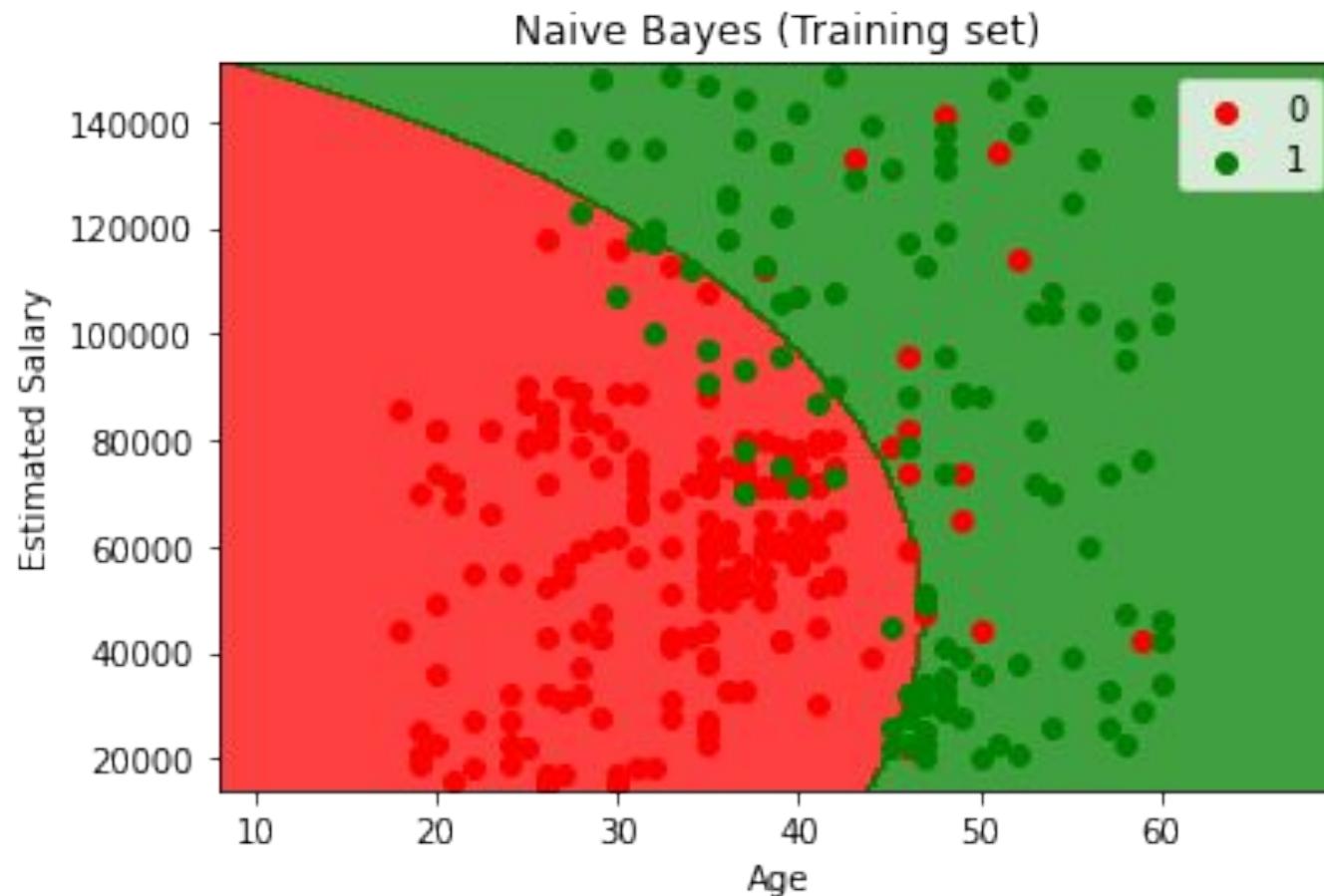
Making the Confusion Matrix

- `from sklearn.metrics import confusion_matrix, accuracy_score`
- `cm = confusion_matrix(y_test, y_pred)`
- `print(cm)`
- `accuracy_score(y_test, y_pred)`
- Output
[[63 3]
 [7 25]]
0.9

Visualising the Training set results

```
• from matplotlib.colors import ListedColormap  
• X_set, y_set = sc.inverse_transform(X_train), y_train  
• X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),  
•                      np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))  
• plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),  
•               alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
• plt.xlim(X1.min(), X1.max())  
• plt.ylim(X2.min(), X2.max())  
• for i, j in enumerate(np.unique(y_set)):  
•     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)  
• plt.title('Naive Bayes (Training set)')  
• plt.xlabel('Age')  
• plt.ylabel('Estimated Salary')  
• plt.legend()  
• plt.show()
```

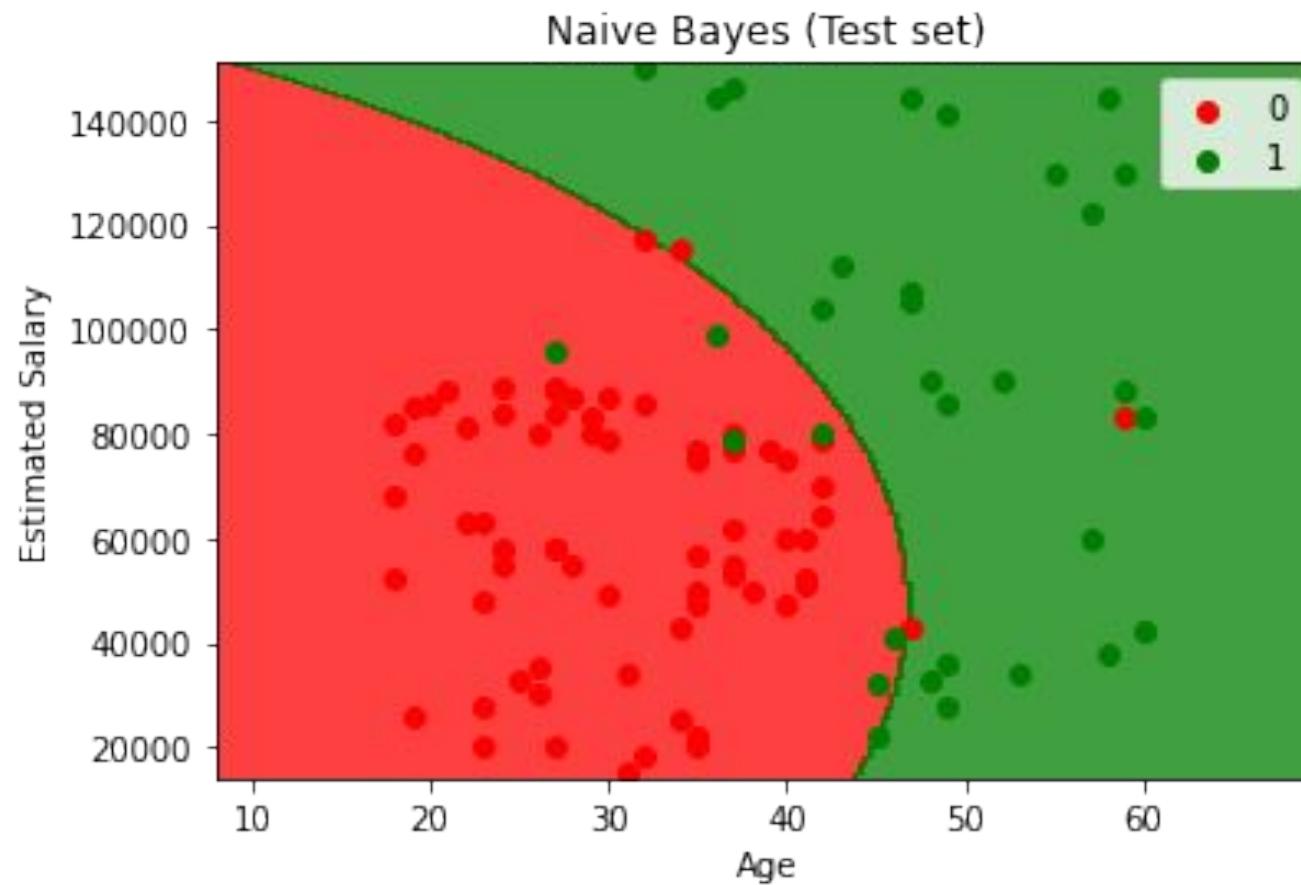
Visualizing the Training set results



Visualizing the Test set results

```
• from matplotlib.colors import ListedColormap  
• X_set, y_set = sc.inverse_transform(X_test), y_test  
• X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),  
•                      np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))  
• plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),  
•               alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
• plt.xlim(X1.min(), X1.max())  
• plt.ylim(X2.min(), X2.max())  
• for i, j in enumerate(np.unique(y_set)):  
•     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)  
• plt.title('Naive Bayes (Test set)')  
• plt.xlabel('Age')  
• plt.ylabel('Estimated Salary')  
• plt.legend()  
• plt.show()
```

Visualising the Test set results



K-Nearest Neighbor

K-Nearest neighbor

- Instance based learning- also called lazy algorithm
 - When we get the training examples we do not process them and learn the model instead we just store the examples
 - And when we need to classify an instant that time we process
 - Algorithm does not come up with the model apriori rather when it gets the test instance it uses the stored instances in memory in order to find the possible model
- Non-linear model
- Very compute intensive
- Decision boundary is a curve
- default parameter for the number of neighbors k
 - 5

How does it do it?

- For a given new instance
 - find what is the closest instance
 - find the y value of that closest instance
 - guess this y values as the y- value of the test instance.
- How to find most similar instance/some neighbouring (most nearest) instances
 - We can use similarity metrics (or distance functions) depending on the type of data that we have
 - Euclidean distance, Cosine and other

1-nearest neighbour

- Training phase
 - save the examples
- Prediction time
 - get the test instance (x_t)
 - find the training example (x_i, y_i) that is closest to test instance (x_t)
 - predict the y_i as the output y_t

K-nearest neighbour

- Training phase- save the examples
- Prediction time- get the test instance (x_t) and find k training example $(x_1, y_1), (x_2, y_2) \dots (x_k, y_k)$ that are closest to test instance (x_t) and we predict
 - the majority class (most frequent) from y_1, y_2, \dots, y_k as the output y'

$$\text{Majority Voting: } y' = \operatorname{argmax}_v \sum_{(x_i, y_i) \in D_s} I(v = y_i),$$

- Where v is the class label, y_i is the class label for one of the nearest neighbours and $I(\cdot)$ is an indicator function that returns the value 1 if its argument is true and 0 otherwise.

Distance-weighted voting scheme

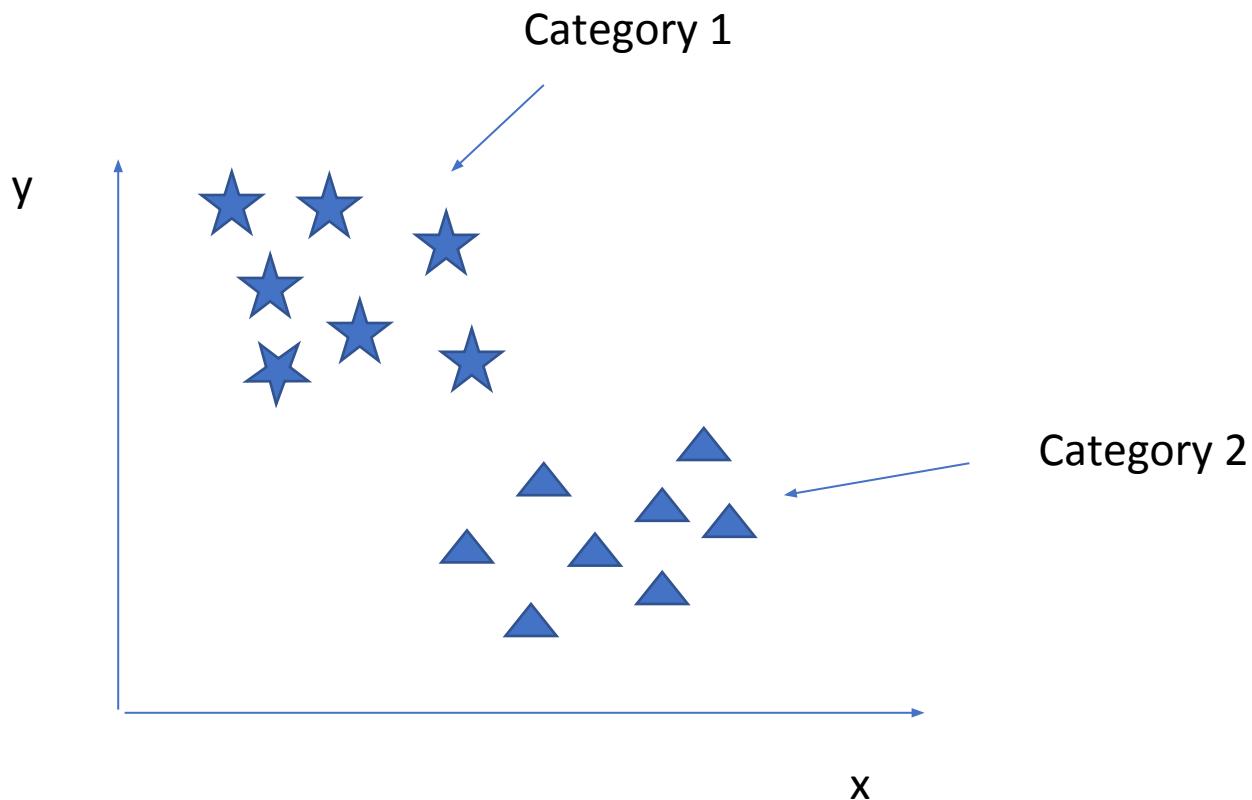
- In the majority voting approach, every neighbor has the same impact on the classification.
- This makes the algorithm sensitive to the choice of k
- One way to reduce the impact of k is to weight the influence of each nearest neighbor x_i according to its distance: $w_i = 1/d(x', x_i)^2$
- As a result, training examples that are located far away from z have a weaker impact on the classification compared to those that are located close to z .
- Using the distance-weighted voting scheme, the class label can be determined as follows:

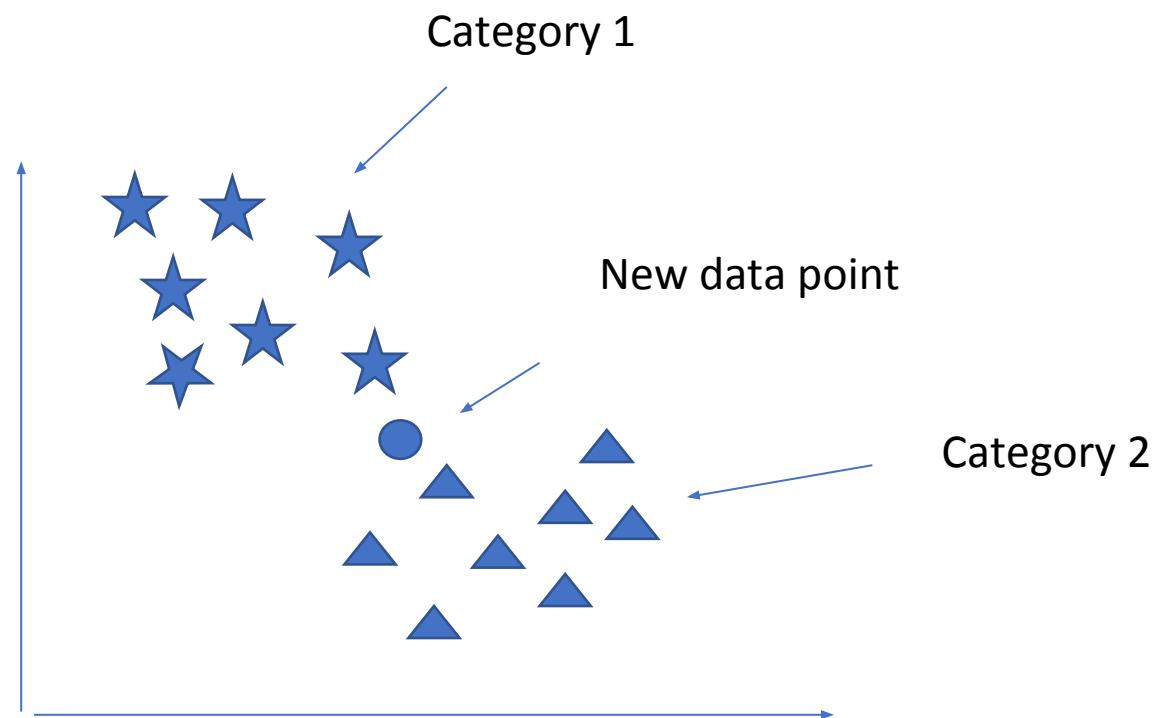
$$\text{Distance-Weighted Voting: } y' = \operatorname{argmax}_v \sum_{(x_i, y_i) \in D_z} w_i \times I(v = y_i).$$

improvements/issues of concern

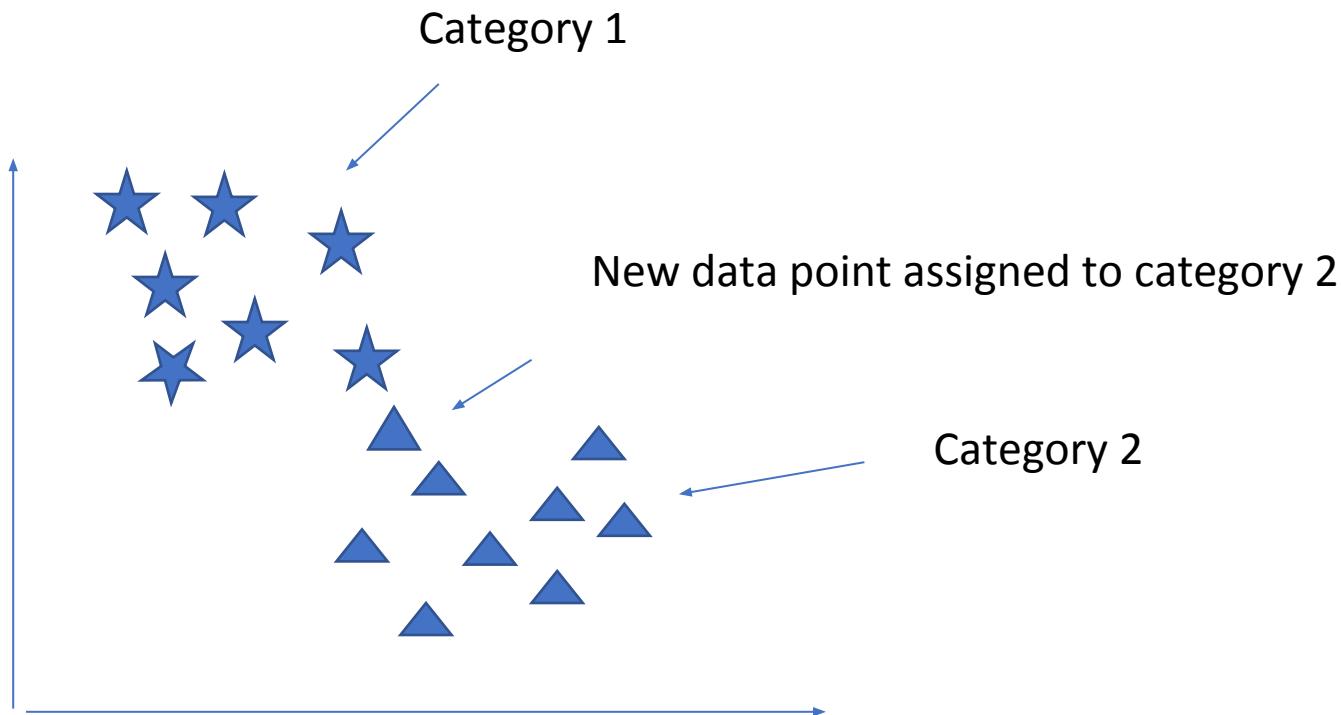
- Weighting examples from the neighbourhood in case of $k > 1$
- How to measure closeness
 - Many distance functions
- How to find the closest points quickly
 - We must use some good data structure or method to store training examples so that we don't need to go through all the training examples, find the distance and come up with the nearest data point

Dataset



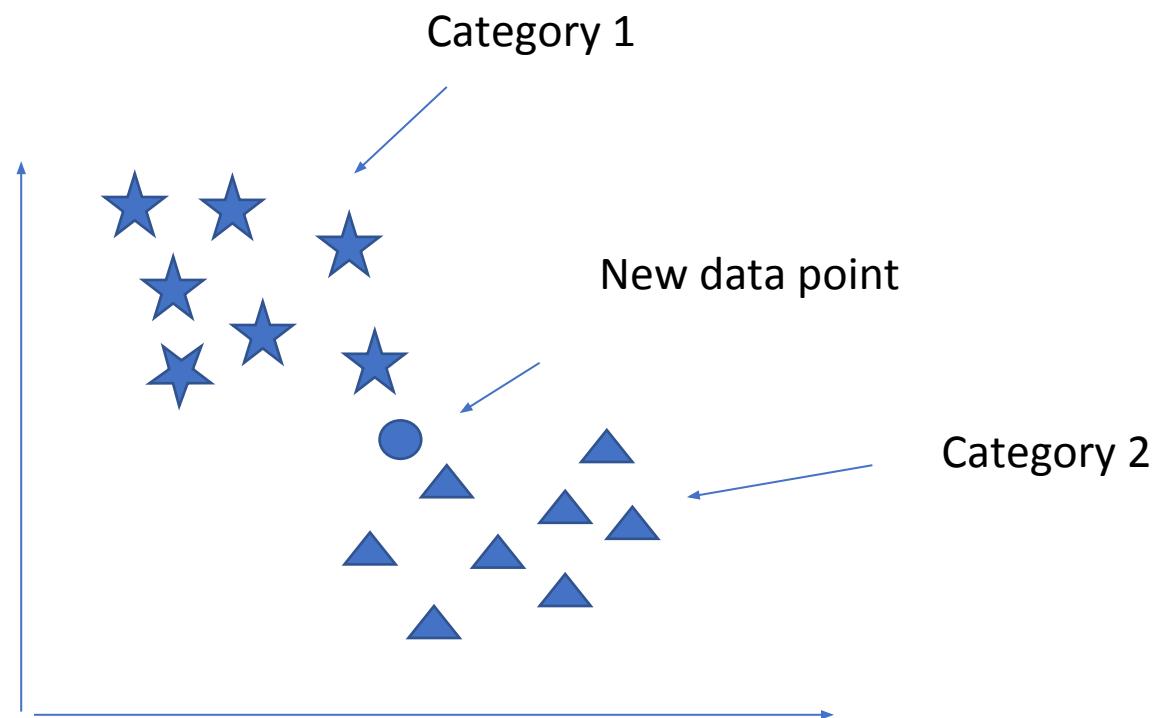


Result of K-NN

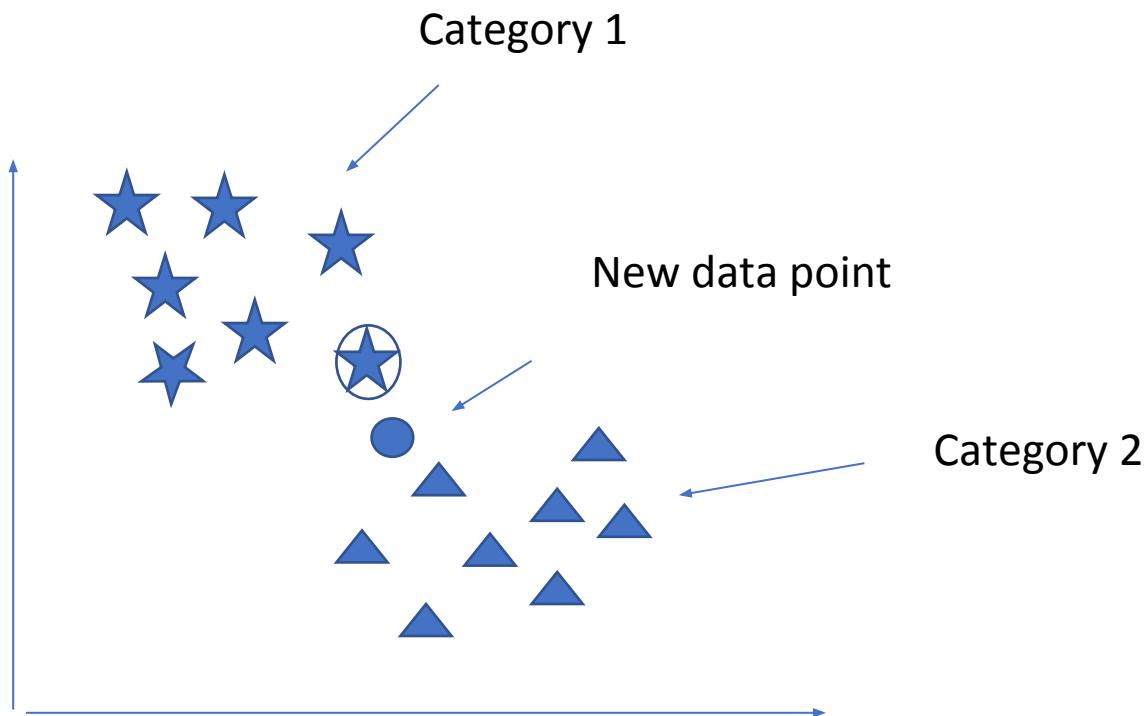


How to prepare model

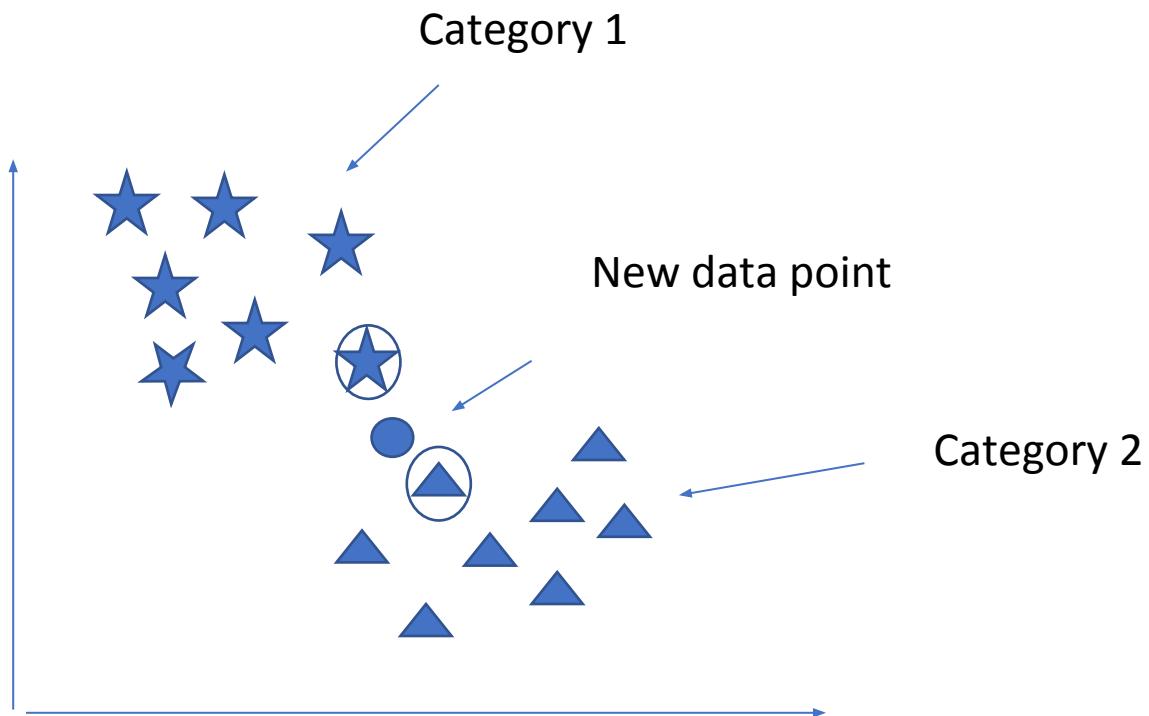
- 1. Choose the number K of neighbors
- 2. Take the K-nearest neighbors of the new data point, according to the Euclidean distance
- 3. Among these K neighbors, count the number of data points in each category
- 4. Assign the new data point to the category where you counted the most neighbors



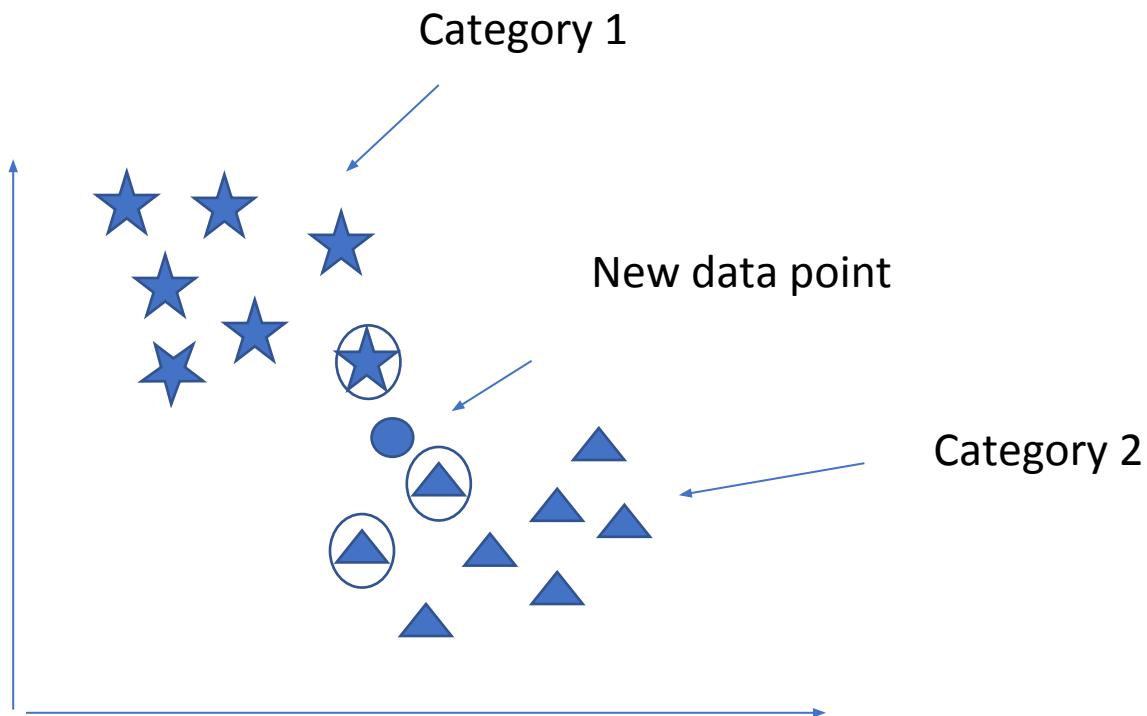
5-NN



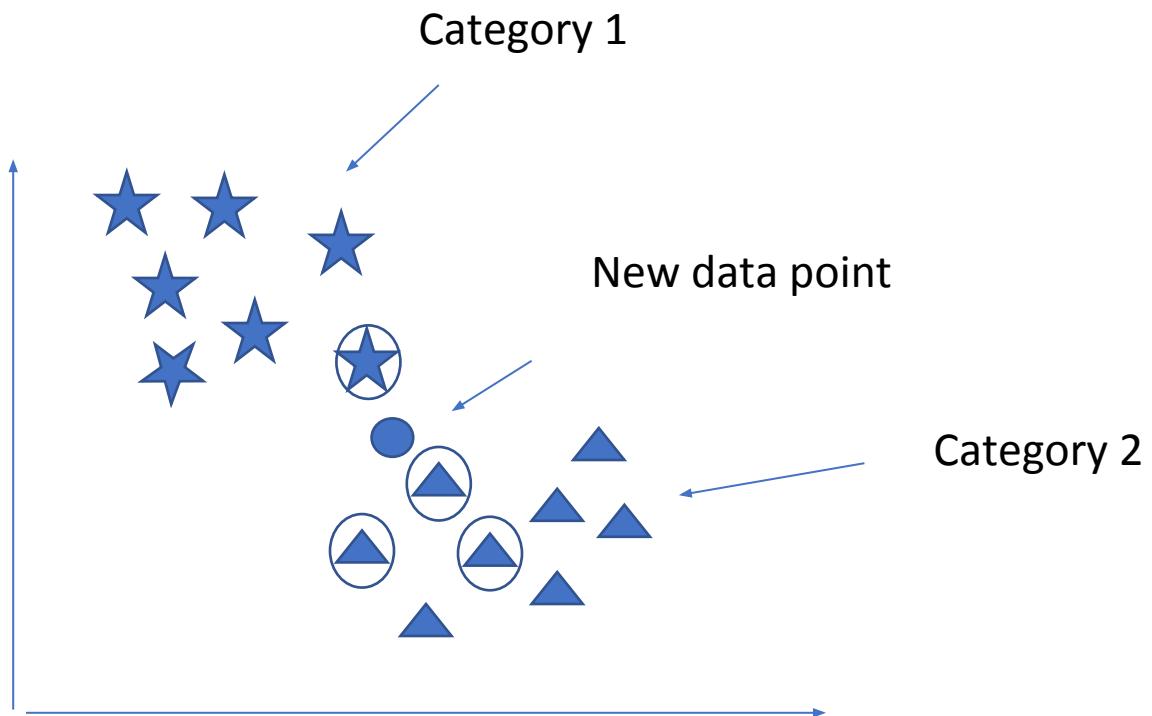
5-NN



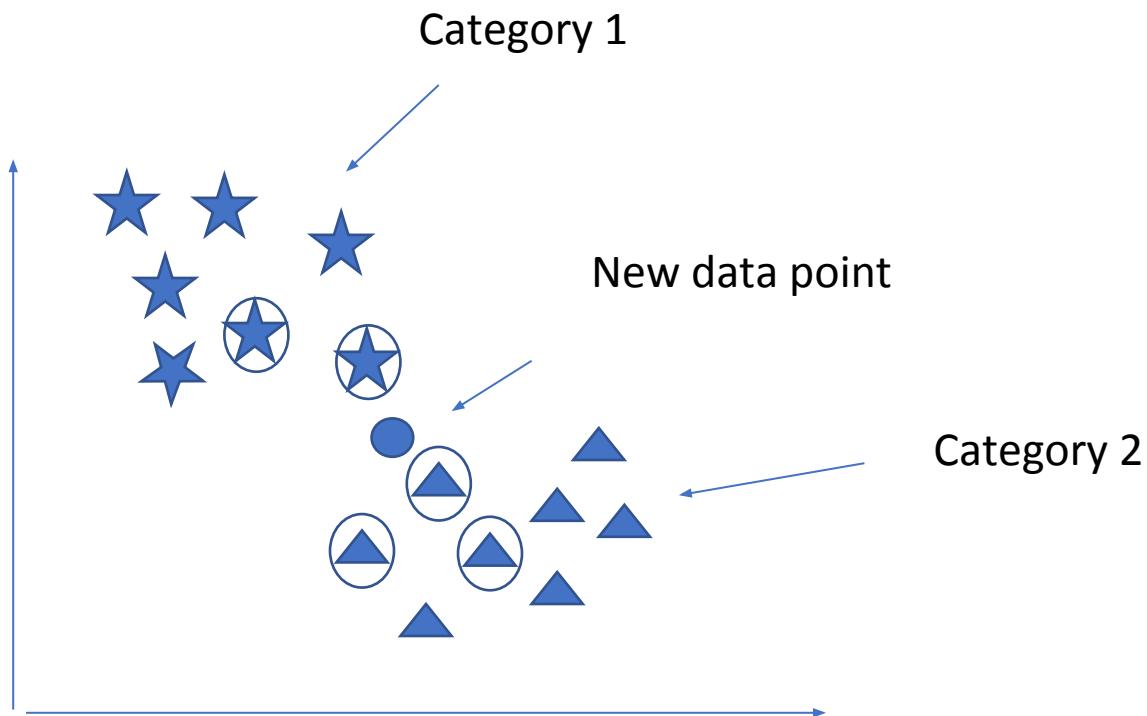
5-NN



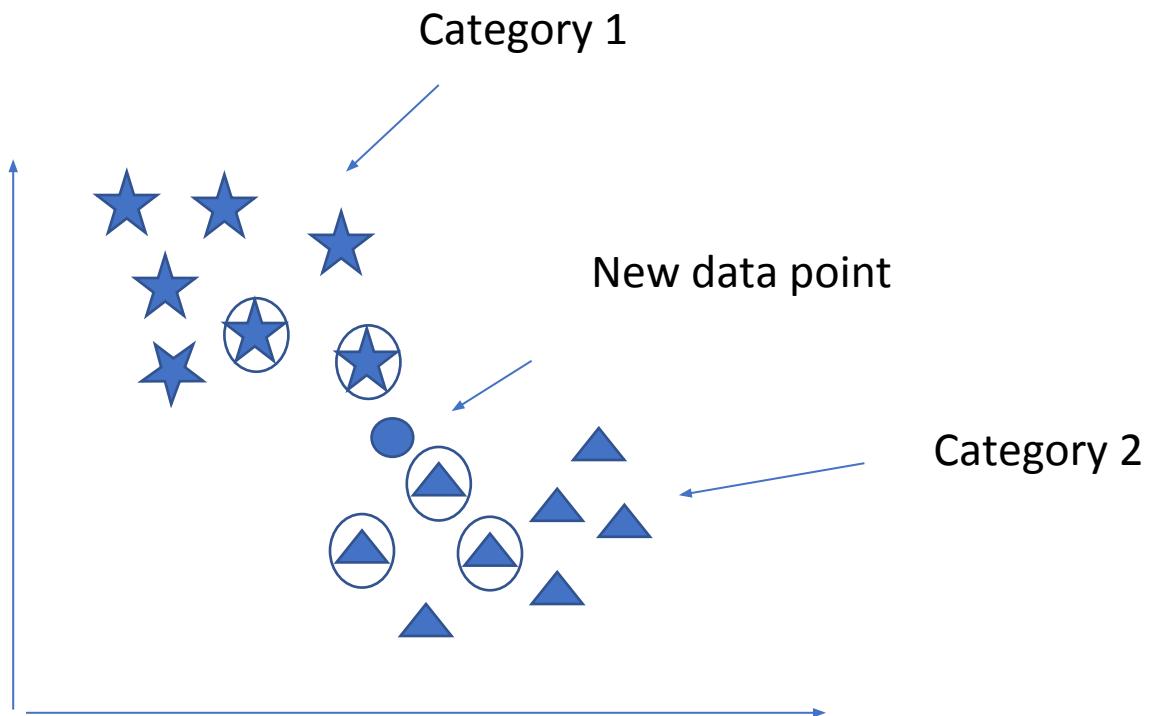
5-NN



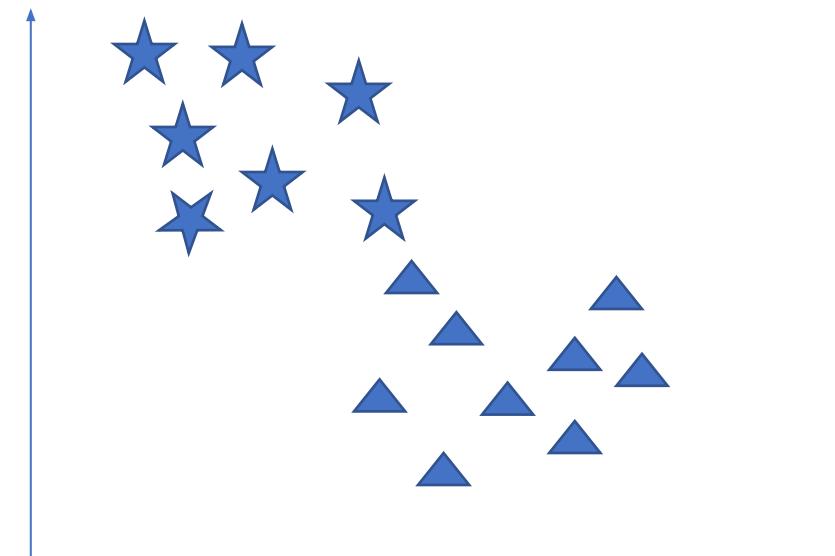
5-NN



5-NN



Category 1- 2 neighbors
Category 2- 3 neighbors



For more than two classes

- For more classes, we count how many neighbors belong to each class and again predict the most common class.

Algorithm

1. Load the data
2. Initialize K to your chosen number of neighbors
3. For each example in the data
 - 3.1 Calculate the distance between the query example and the current example from the data.
 - 3.2 Add the distance and the index of the example to an ordered collection
4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
5. Pick the first K entries from the sorted collection
6. Get the labels of the selected K entries
7. If regression, return the mean of the K labels
8. If classification, return the mode of the K labels

- In cases where we are taking a majority vote (e.g. picking the mode in a classification problem) among labels, we usually make K an odd number to have a tiebreaker.
- for non-binary classification k can have any even or odd value

Importing the libraries

```
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
```

```
x = dataset.iloc[:, :-1].values
```

```
y = dataset.iloc[:, -1].values
```

	Age	EstimatedSalary	Purchased
	19	19000	0
	35	20000	0
	26	43000	0
	27	57000	0
	19	76000	0
	27	58000	0
	27	84000	0
	32	150000	1
	25	33000	0

Total- 1000 entries

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split  
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
x_train = sc.fit_transform(x_train)  
x_test = sc.transform(x_test)
```

Training the K-NN model on the Training set

```
from sklearn.neighbors import KNeighborsClassifier  
classifier = KNeighborsClassifier(n_neighbors = 5, metric = minkowski,  
p =2)  
classifier.fit(X_train, y_train)
```

Predicting a new result

- `print(classifier.predict(sc.transform([[30,87000]])))`
- Output
 - [0]

Predicting the Test set results

```
y_pred = classifier.predict(X_test)  
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

Making the Confusion Matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score  
cm = confusion_matrix(y_test, y_pred)  
print(cm)  
accuracy_score(y_test, y_pred)
```

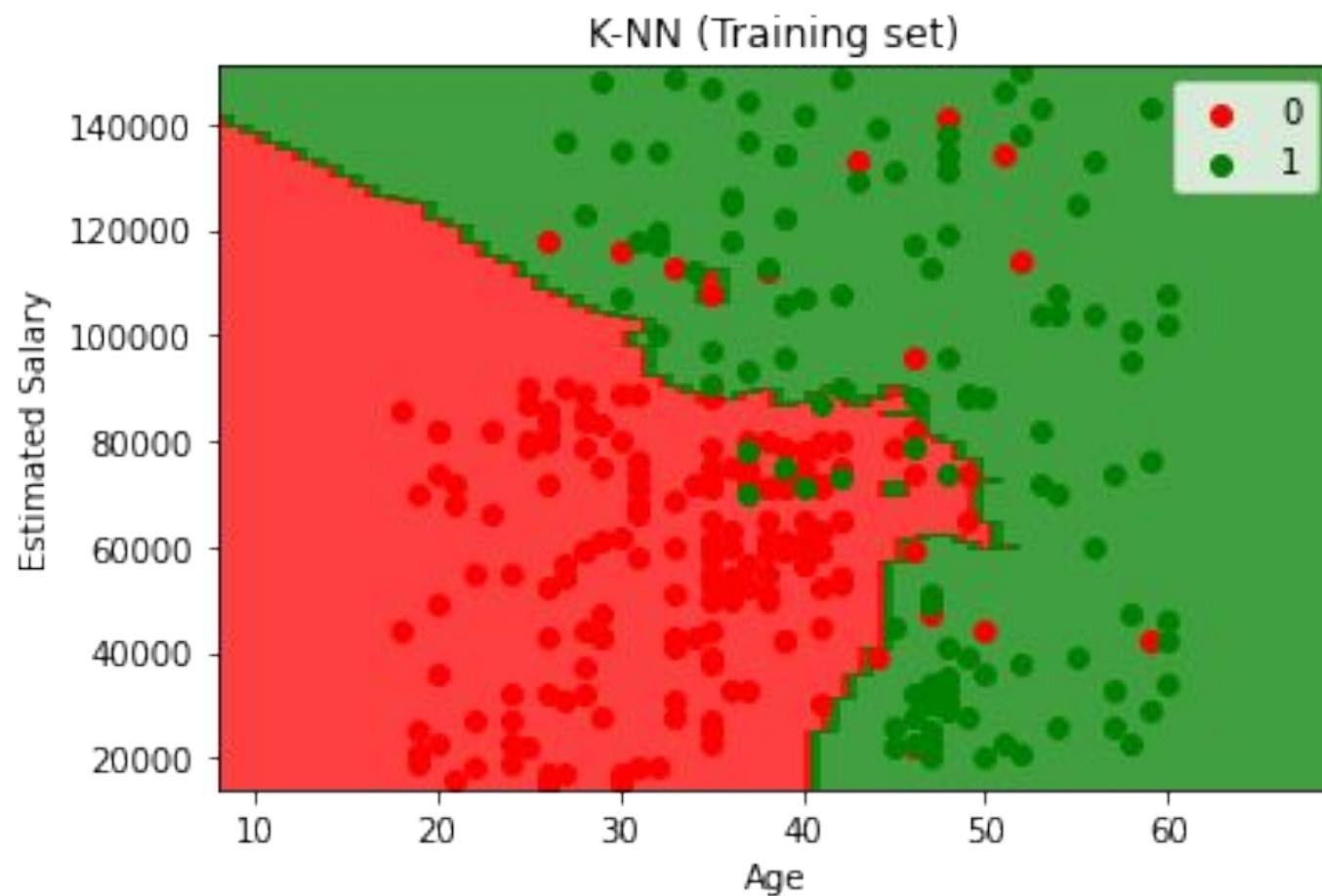
Confusion Matrix

		Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)	
	False Negatives (FNs)	True Negatives (TNs)	
Predicted Negative (0)			

Visualising the Training set results

- from matplotlib.colors import ListedColormap
- X_set, y_set = sc.inverse_transform(X_train), y_train
- X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 1),
 np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 1))
- plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape)
,
- alpha = 0.75, cmap = ListedColormap(('red', 'green')))
- plt.xlim(X1.min(), X1.max())
- plt.ylim(X2.min(), X2.max())
- for i, j in enumerate(np.unique(y_set)):
- plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
- plt.title('K-NN (Training set)')
- plt.xlabel('Age')
- plt.ylabel('Estimated Salary')
- plt.legend()
- plt.show()

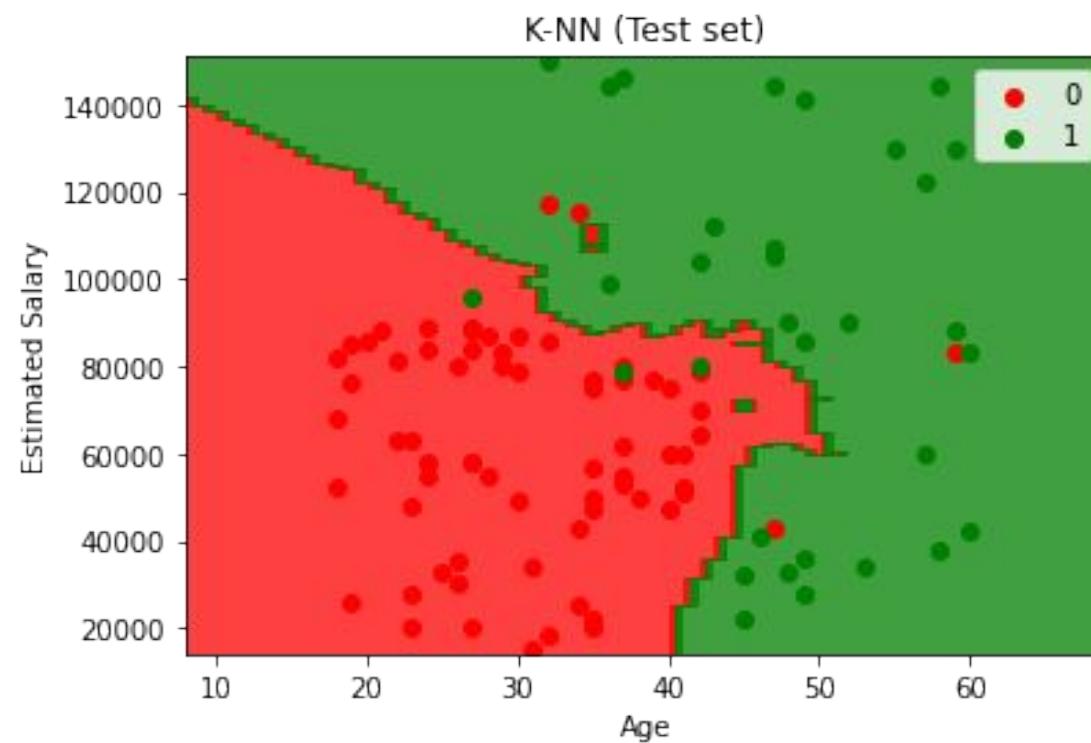
Output



Visualising the Test set results

- from matplotlib.colors import ListedColormap
- X_set, y_set = sc.inverse_transform(X_test), y_test
- X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 1),
• np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 1))
- plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
• alpha = 0.75, cmap = ListedColormap(('red', 'green')))
- plt.xlim(X1.min(), X1.max())
- plt.ylim(X2.min(), X2.max())
- for i, j in enumerate(np.unique(y_set)):
- plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
- plt.title('K-NN (Test set)')
- plt.xlabel('Age')
- plt.ylabel('Estimated Salary')
- plt.legend()
- plt.show()

Output



Decision Tree Classification

Decision Trees

- Classification trees
- Regression trees

- The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by **learning simple decision rules** inferred from prior data (training data).
- 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**
- The decisions or the test are performed on the basis of features of the given dataset
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees

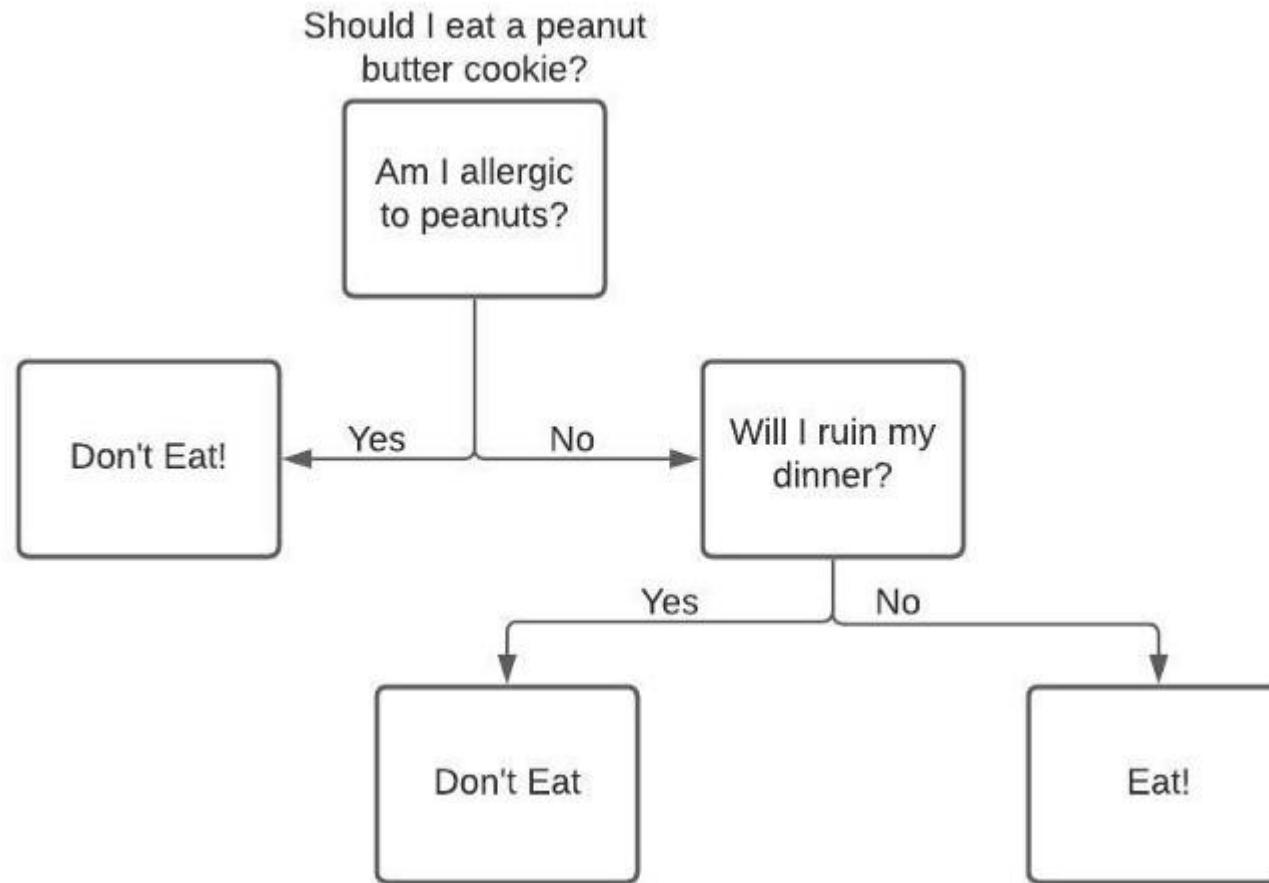
Decision Trees

- They were popular around 25 years ago but then more sophisticated methods replaced them
- Recently they were reborn with new upgrades
 - Additional methods that build on top of decision trees
 - Random forests, gradient boosting etc.
- It is quite a simple method but it lies in the foundation of some of the more modern and powerful methods
- The family of decision tree learning algorithms includes algorithms like
 - ID3, CART, ASSISTANT, etc.

Decision tree

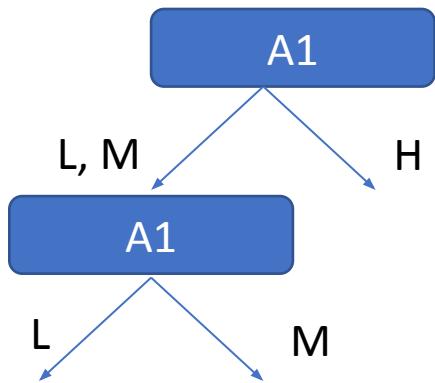
- Has two types of nodes
 - Decision nodes
 - Specify a test based on which we decide which branch we should follow
 - This test is usually done on the value of a feature of the instance
 - Leaf nodes
 - Indicate the classification of an example or the value of the example
 - it can be the predicted value of the example for classification or regression or it can be a probability

Decision tree Example



- 2 decision nodes
- 3 leaf nodes

- We could have more than 2 children also for example if attribute has low, medium and high values then it will have three children
- In case of multivalued attribute same attribute can be used again to split



Which decision tree we should choose

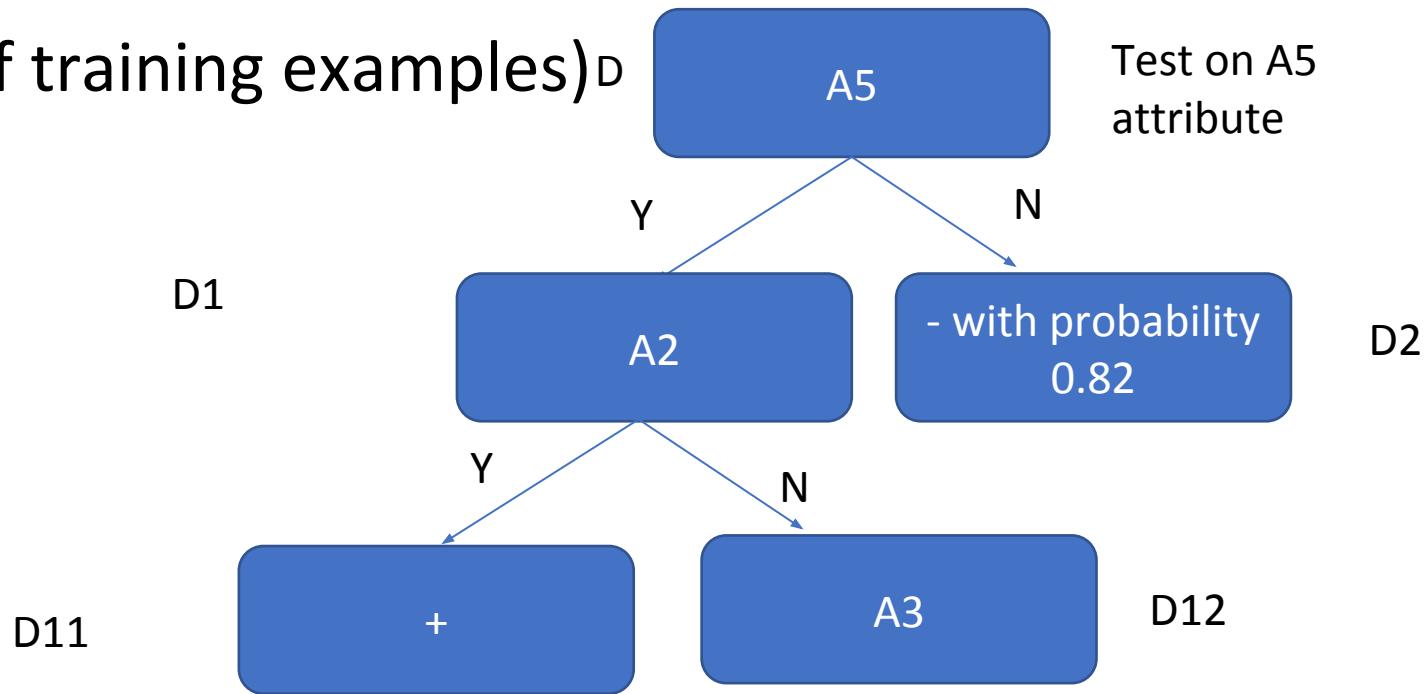
- For the given training examples we have to generate decision tree
- Given the training examples there is possibility
 - we can get many decision trees that fit the training examples
 - Which one we should choose
 - Or if the examples are noisy it could be that there is no decision which exactly fit the data
 - Which tree we should choose that has low error
- Bias
 - Using bias we restrict the hypothesis space or we put preferences on the hypothesise space
- Once we have chosen decision tree as the hypothesis space we can put some preference
 - Commonly preference for decision tree is to have tree with smaller trees (low depth or small no of nodes)

Come up with the algorithm that

- Will search the space of the decision trees and given the training examples come up with the smallest decision tree that fits the data (if there is no noise).
 - But finding the smallest decision tree that fits the data is a computationally hard problem therefore we look for some greedy algorithm
 - We search for a good tree and we have to decide how we can come up with the good tree for learning

How the decision tree is build?

- D (a set of training examples)



So we can grow the tree so that every leaf has a specific value or we stop at a point where at a leaf there are more than one possible values but one is more dominant.

- The space of decision tree is too big for systematic search
- Learning problem is given a set of decision trees we have to find a good tree
 - Two choices that we need to make at a time
 - Stop
 - Return a value for the target feature
 - Stopping criteria
 - If all the attributes are exhausted
 - If all the examples are positive/negative
 - If we get too few examples (say 2) in the node
 - Continue
 - Choose a test or choose an attribute to continue with

Top Down Induction of Decision trees ID3

- 1. $A \leftarrow$ the best decision attribute for the next node
- 2. Assign A as decision attribute for node
- 3. for each value of A create new descendent
- 4. sort training examples to leaf node according to the attribute value of the branch
- 5. if all training examples are perfectly classified (same value of target attribute) stop, else iterate over new leaf nodes.
- Choices that we have to take in this algorithm
 - When we have the partial decision tree, which node to continue?
 - Which attribute to use for the test?
 - When to stop?

Advantages

- 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

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
- Decision Node
 - When a sub-node splits into further sub-nodes, then it is called the decision node
- 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
 - When we remove sub-nodes of a decision node, this process is called pruning
 - It is just the opposite process of splitting
- Parent/Child node
 - The root node of the tree is called the parent node, and other nodes are called the child nodes

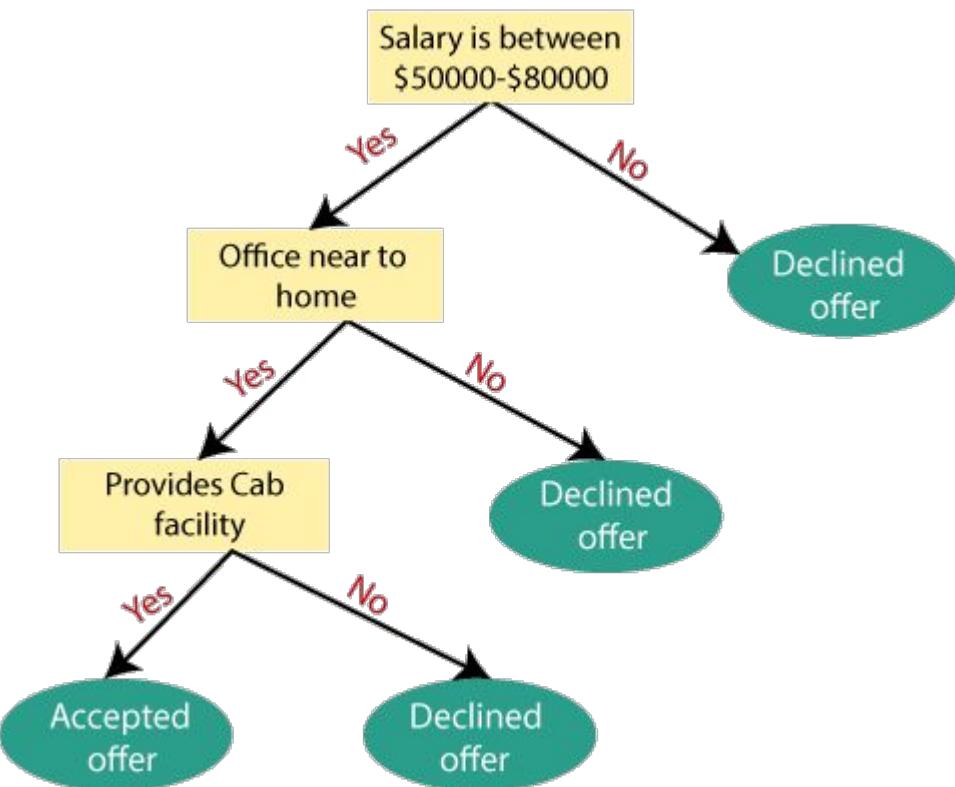
Working

- **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

How to predicting the class of the given record

- Start from the root node of the tree
- This algorithm compares the values of root attribute with the record 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

Example 1

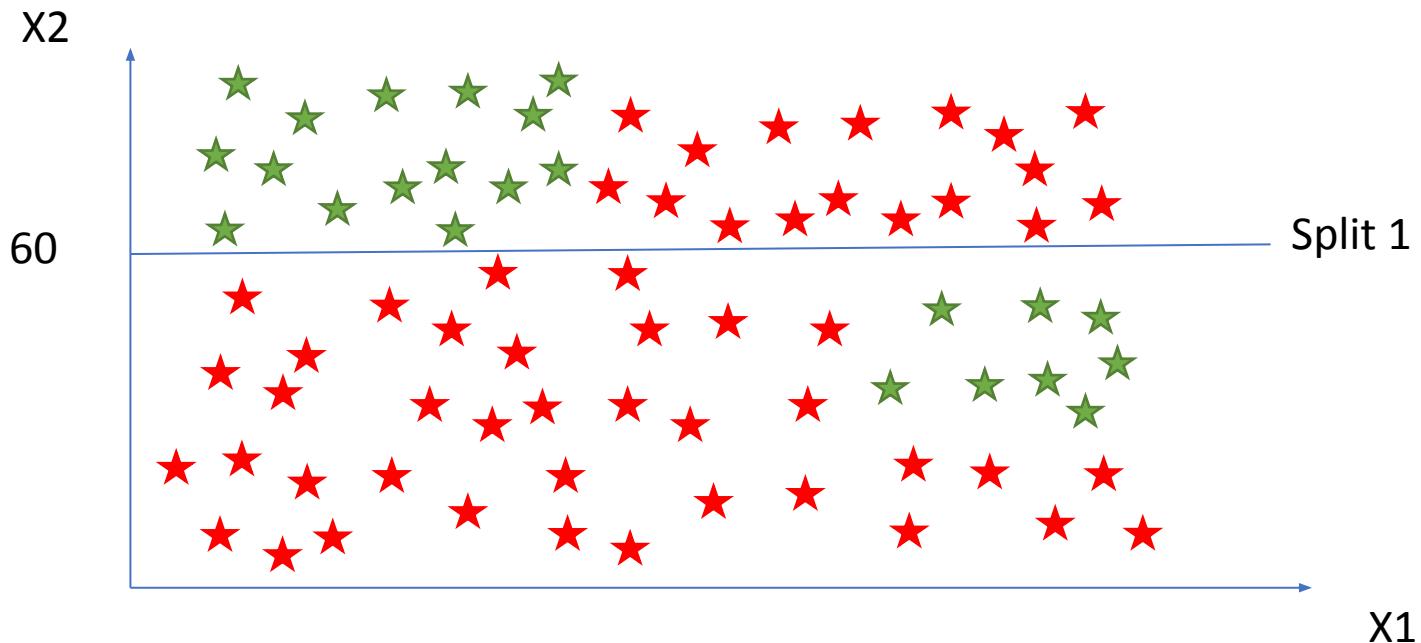


- Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not.
- To solve this problem, the decision tree starts with the root node (Salary attribute by ASM).
- The root node splits further into the next decision node (distance from the office) and one 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).

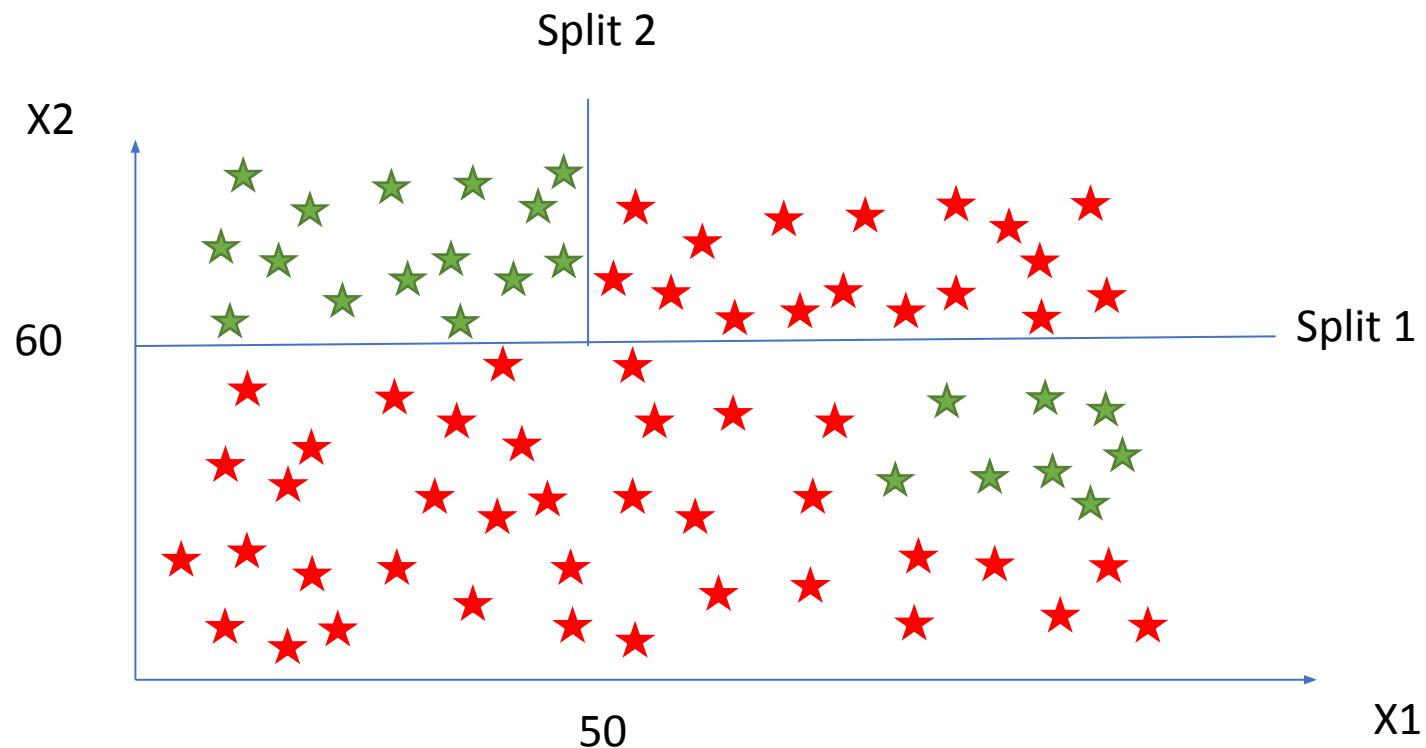
Example 2



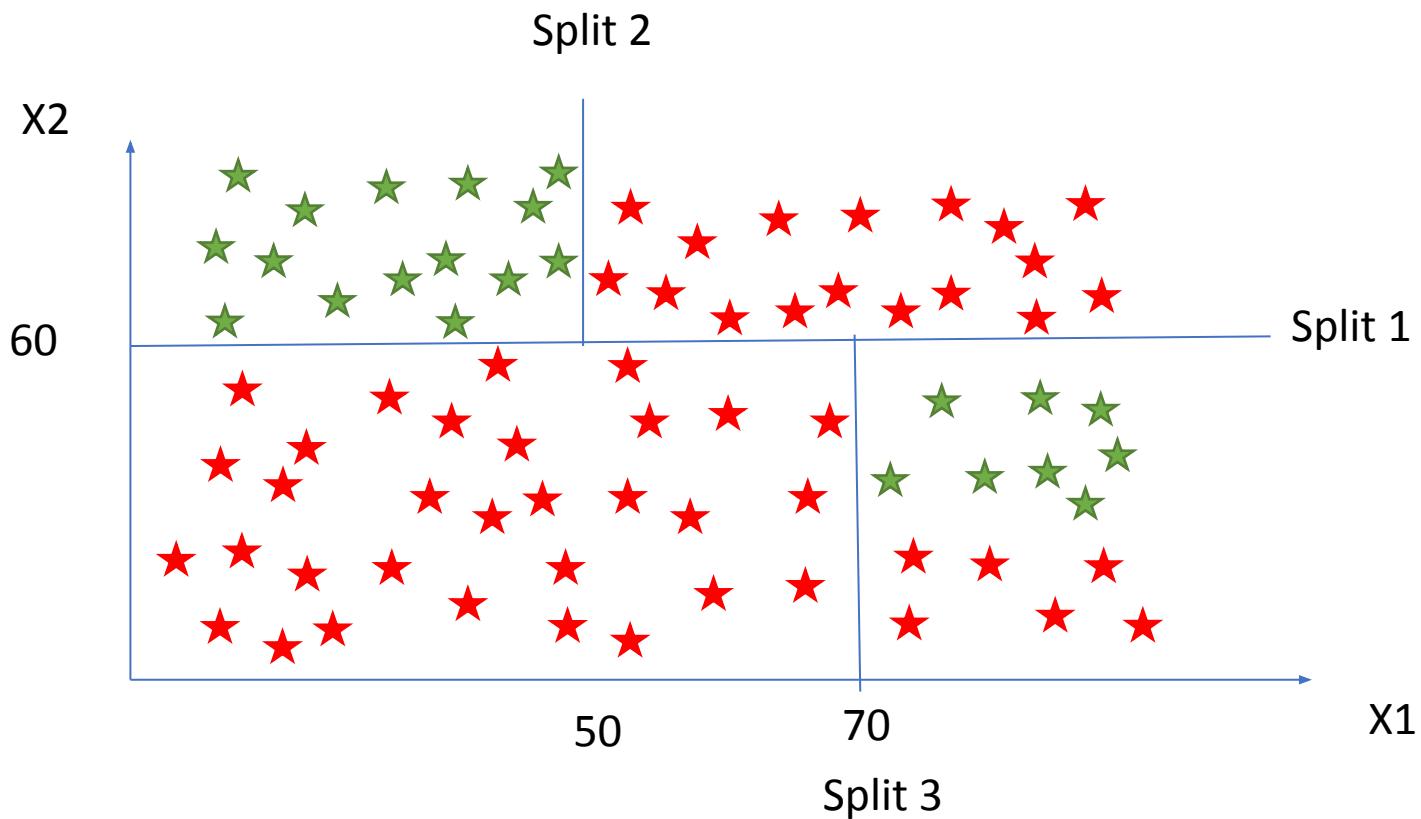
Decision Tree



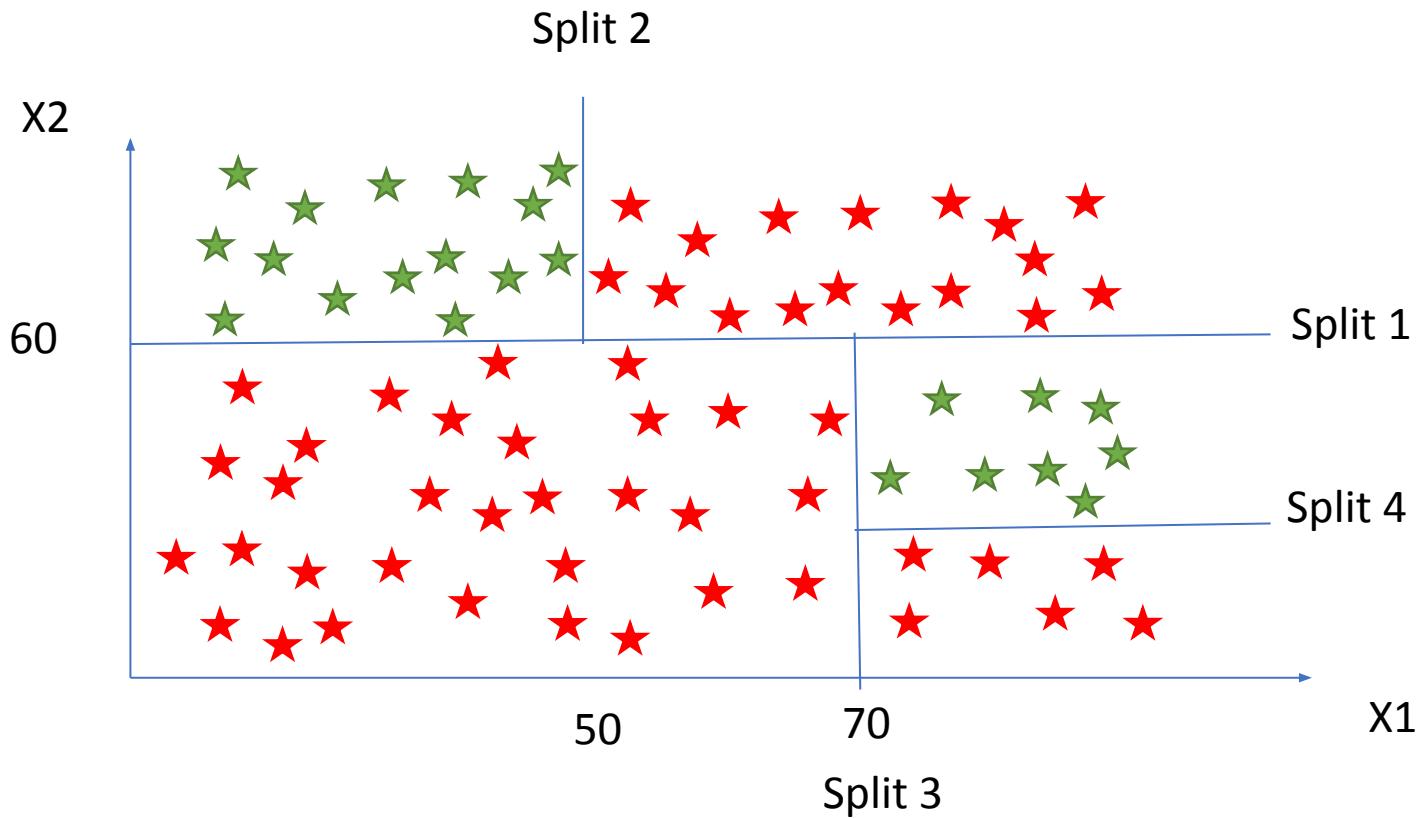
Decision Tree



Decision Tree



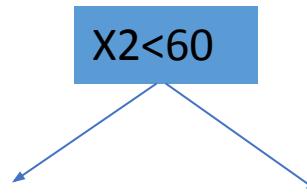
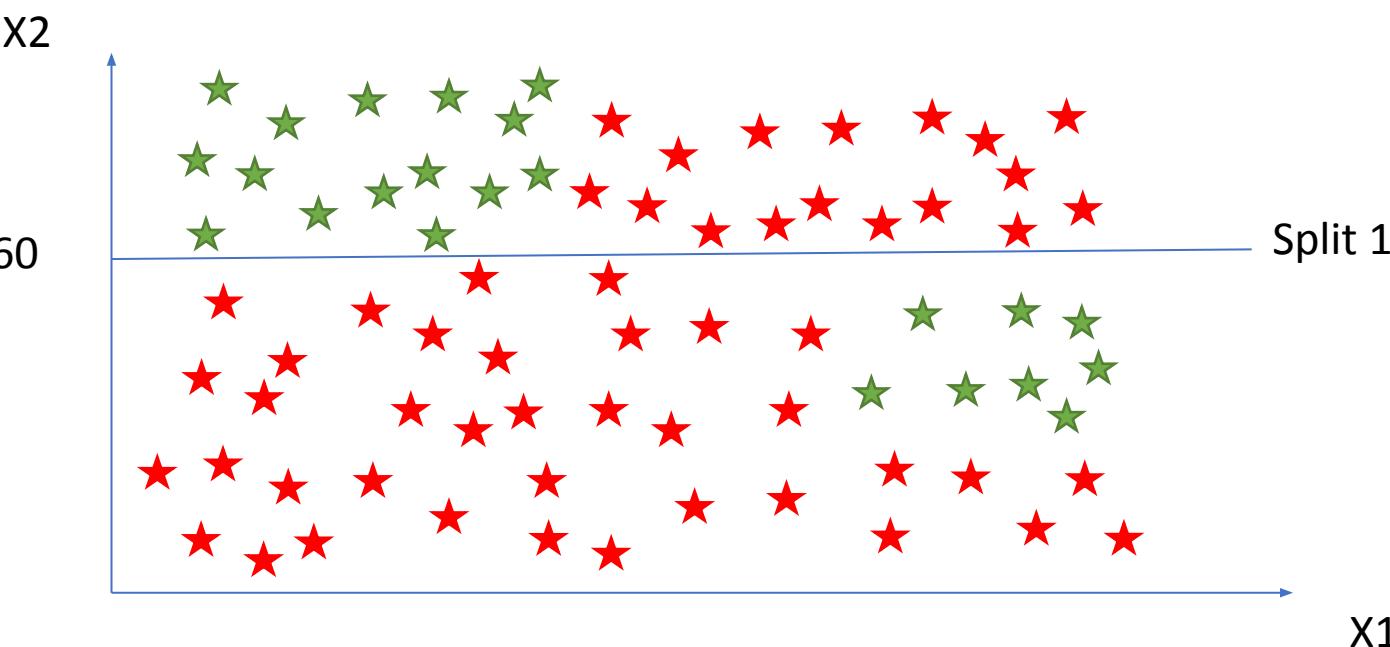
Decision Tree



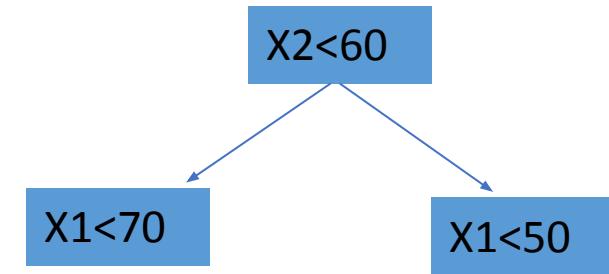
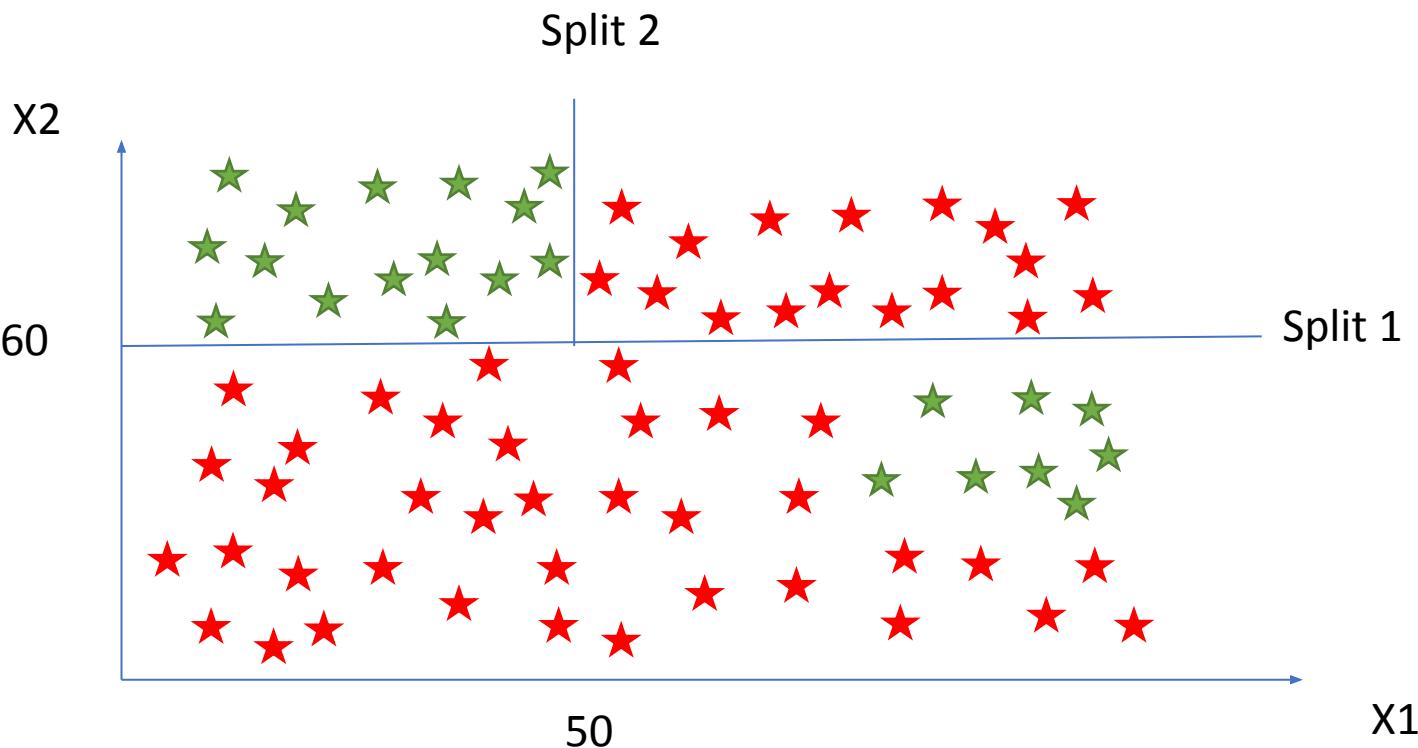
Decision tree Intuition

- Split is trying to minimize entropy
- Optimal splits maximize number of different points in each one of these new leaf

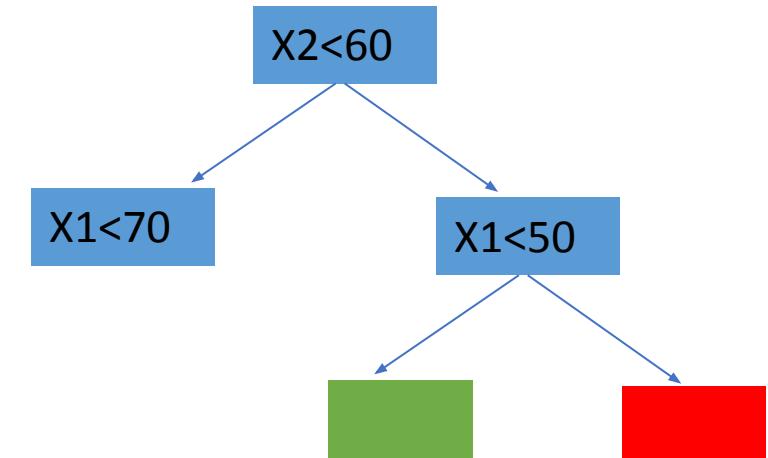
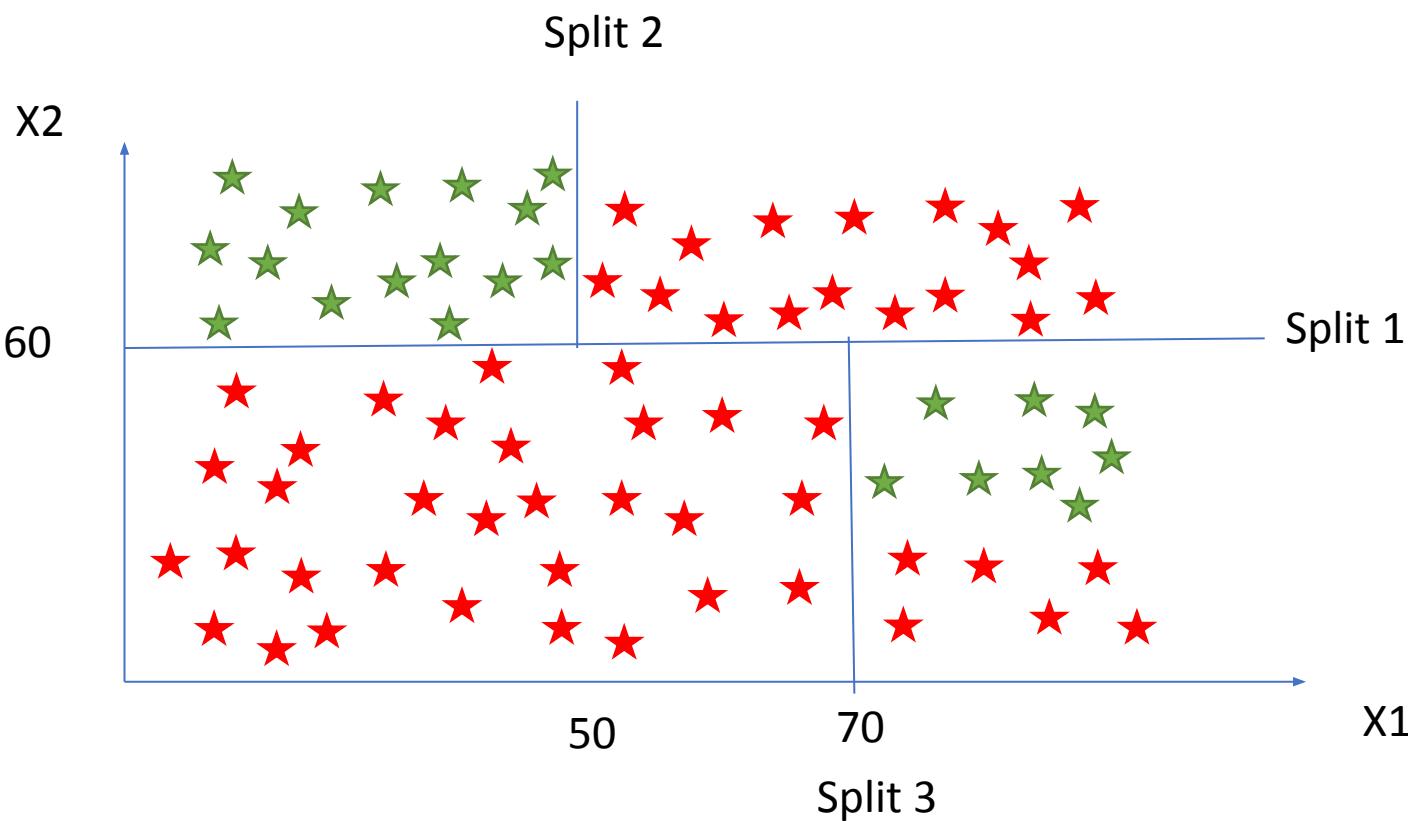
Decision Tree



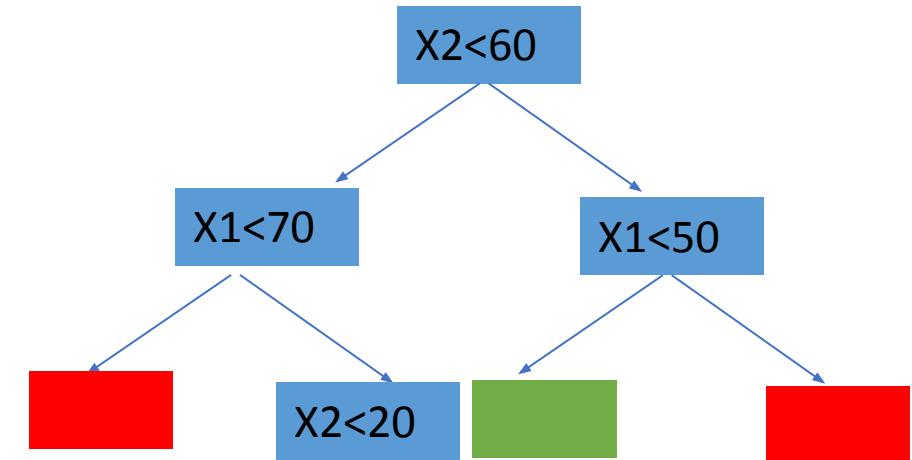
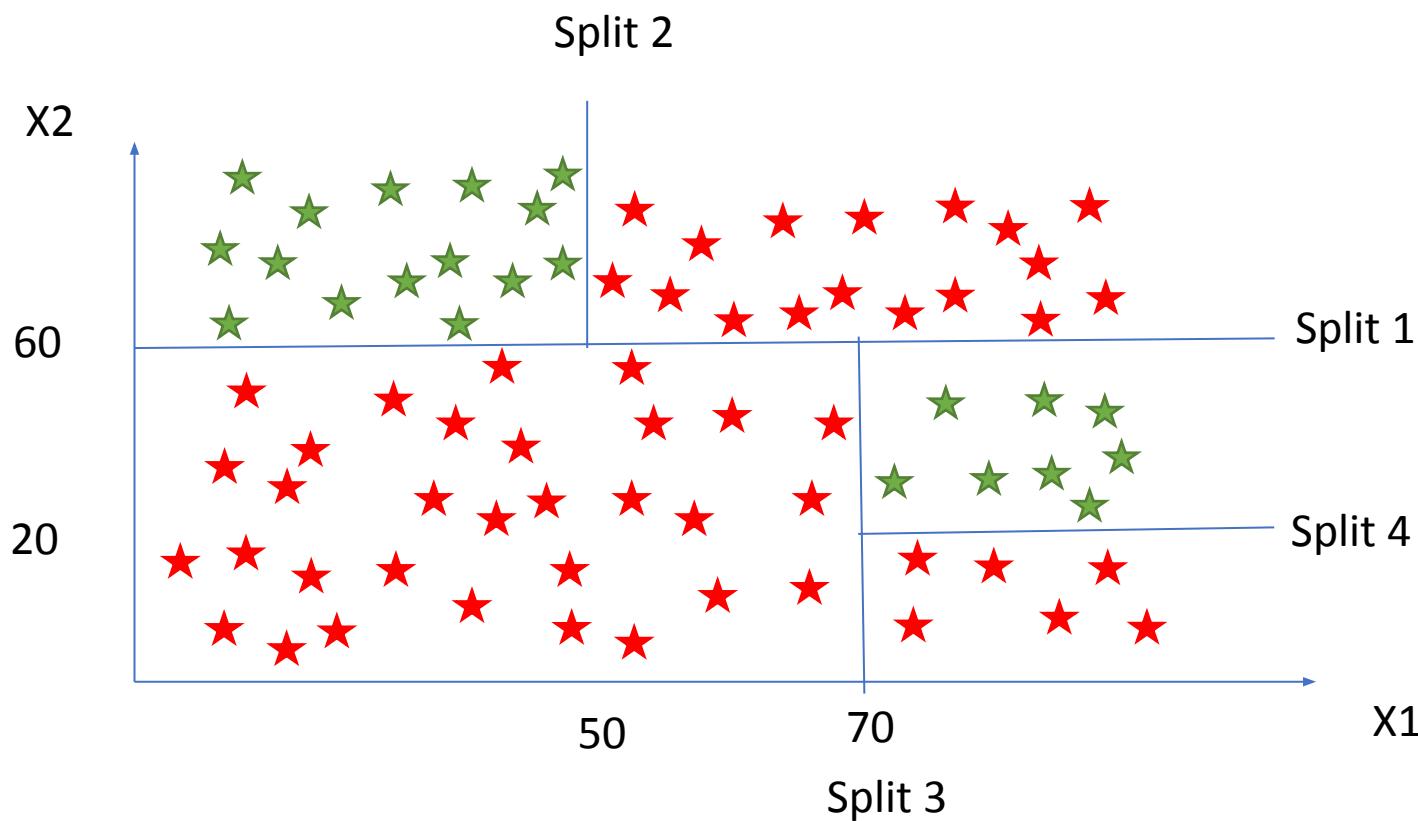
Decision Tree



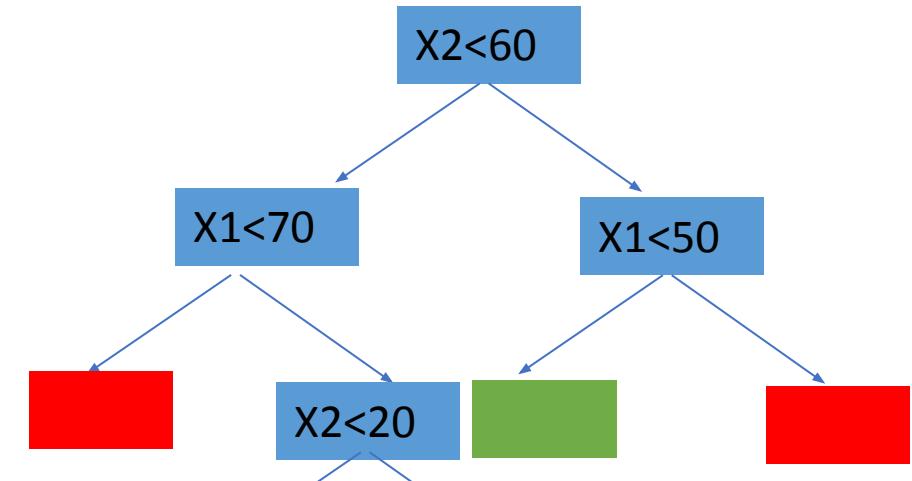
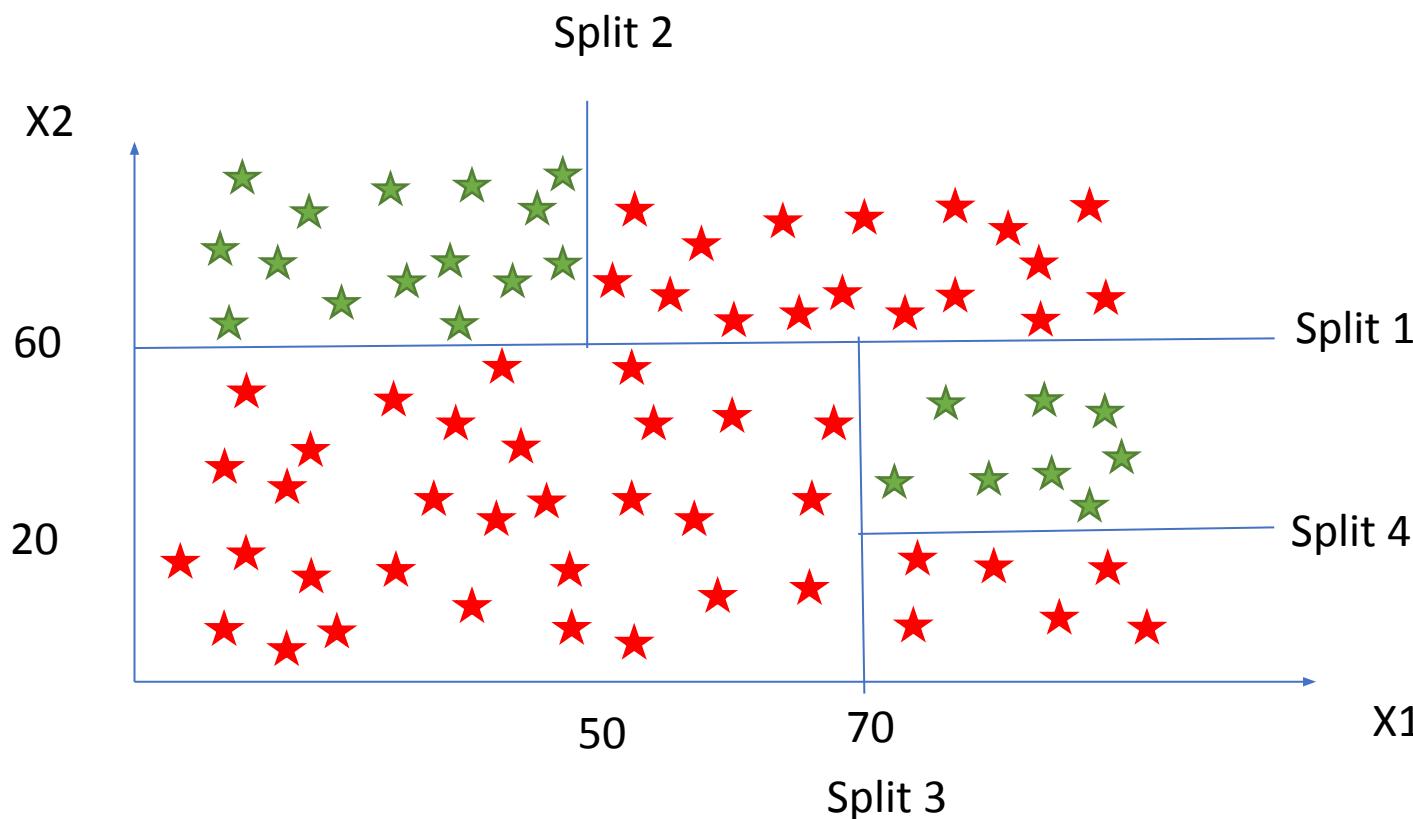
Decision Tree



Decision Tree



Decision Tree



In case of continuous data

- Decision tree has difficulty to process continuous data.

SOP

- Decision tree can be expressed as a Boolean function which is a disjunction of conjunction
- Decision Trees follow **Sum of Product (SOP)** representation
- The Sum of product (SOP) is also known as **Disjunctive Normal Form**
- For a class, every branch from the root of the tree to a leaf node having the same class is conjunction (product) of values, different branches ending in that class form a disjunction (sum)

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
- To solve such problems **Attribute selection measure (ASM) is used**
 - By this measurement, we can easily select the best attribute for the nodes of the tree
- Which attribute to split on
 - Split gives smallest error

How to Split

- Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes
- The creation of sub-nodes increases the homogeneity of resultant sub-nodes
- In other words, we can say that the purity of the node increases with respect to the target variable
- The decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes

- There are two popular techniques for ASM, which are:
 - **Information Gain**
 - **Gini Index**
1. In case of Continuous Target Variable
 1. Reduction in Variance
 2. In case of Categorical Target Variable
 1. Gini Impurity
 2. Information Gain
 3. Chi-Square

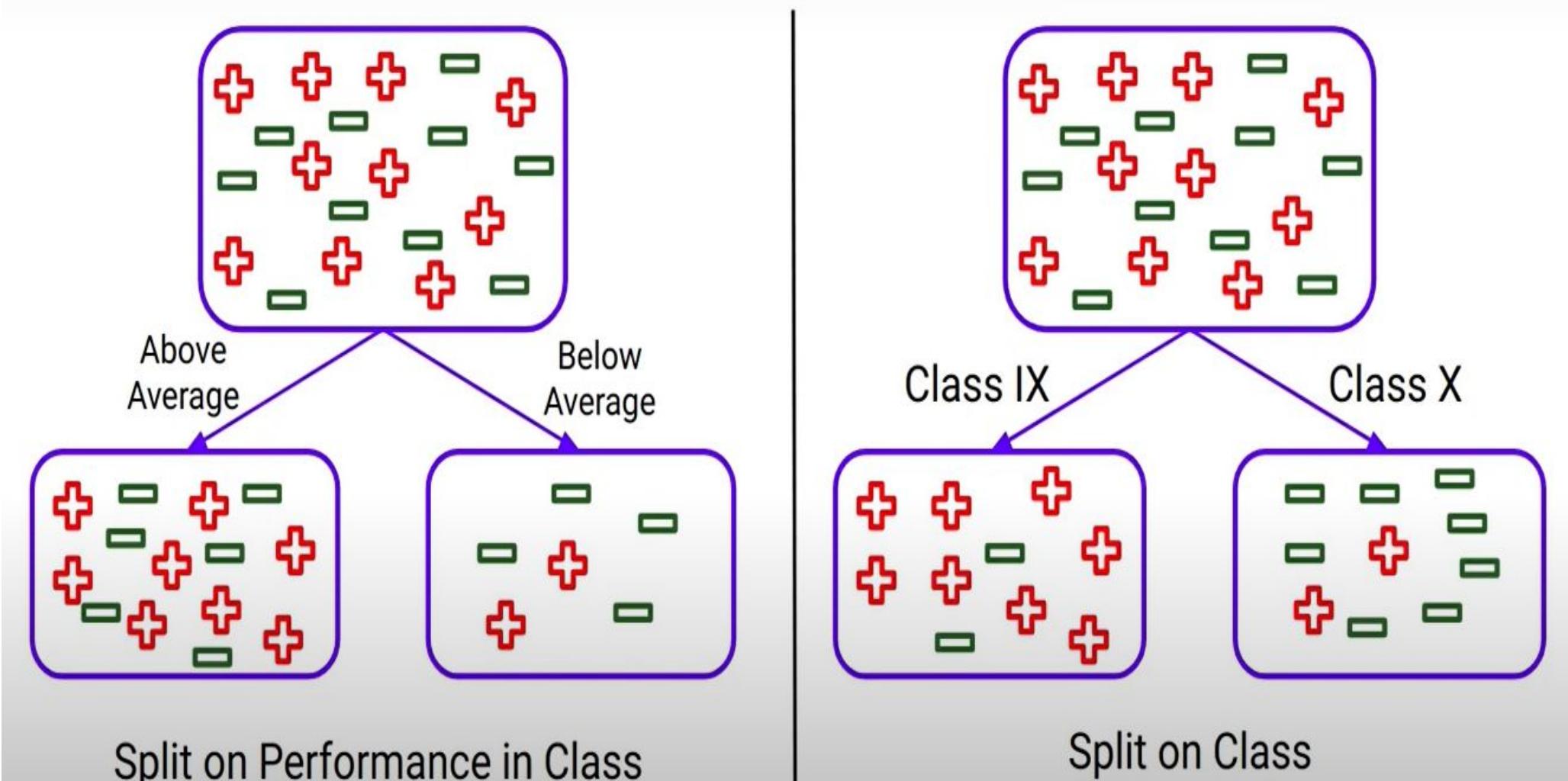
Decision Tree Splitting: Reduction in Variance

- Reduction in Variance is a method for splitting the node used when the target variable is continuous, i.e., regression problems.
- it uses variance as a measure for deciding the feature on which node is split into child nodes.

$$Variance = \frac{\sum (X - \mu)^2}{N}$$

Decision Tree Splitting: Reduction in Variance

- Variance is used for calculating the homogeneity of a node. If a node is entirely homogeneous, then the variance is zero.
- Steps to split a decision tree using reduction in variance:
 1. For each split, individually calculate the variance of each child node
 2. Calculate the variance of each split as the weighted average variance of each child nodes
 - Select the attribute for split having minimum split variance
 3. Perform steps 1-3 until completely homogeneous nodes are achieved



- Class IX node:

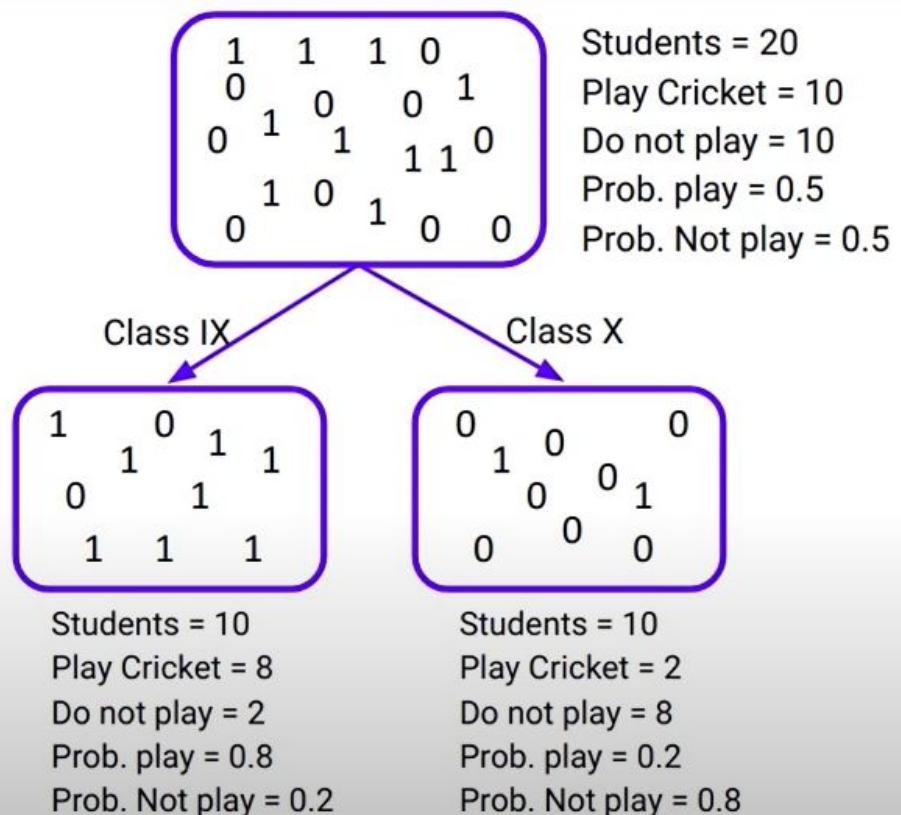
- Mean = $(8*1 + 2*0) / 10 = 0.8$
- Variance =
 $[8*(1-0.8)^2 + 2*(0-0.8)^2] / 10 = 0.16$

- Class X node:

- Mean = $(2*1 + 8*0) / 10 = 0.2$
- Variance =
 $[2*(1-0.2)^2 + 8*(0-0.2)^2] / 10 = 0.16$

- Variance: Class:

$$(10/20)*0.16 + (10/20)*0.16 = 0.16$$



- Above Average node:

- Mean = $(8*1 + 6*0) / 14 = 0.57$

- Variance =

$$[8*(1-0.57)^2 + 6*(0-0.57)^2] / 14 = 0.245$$

- Below Average node:

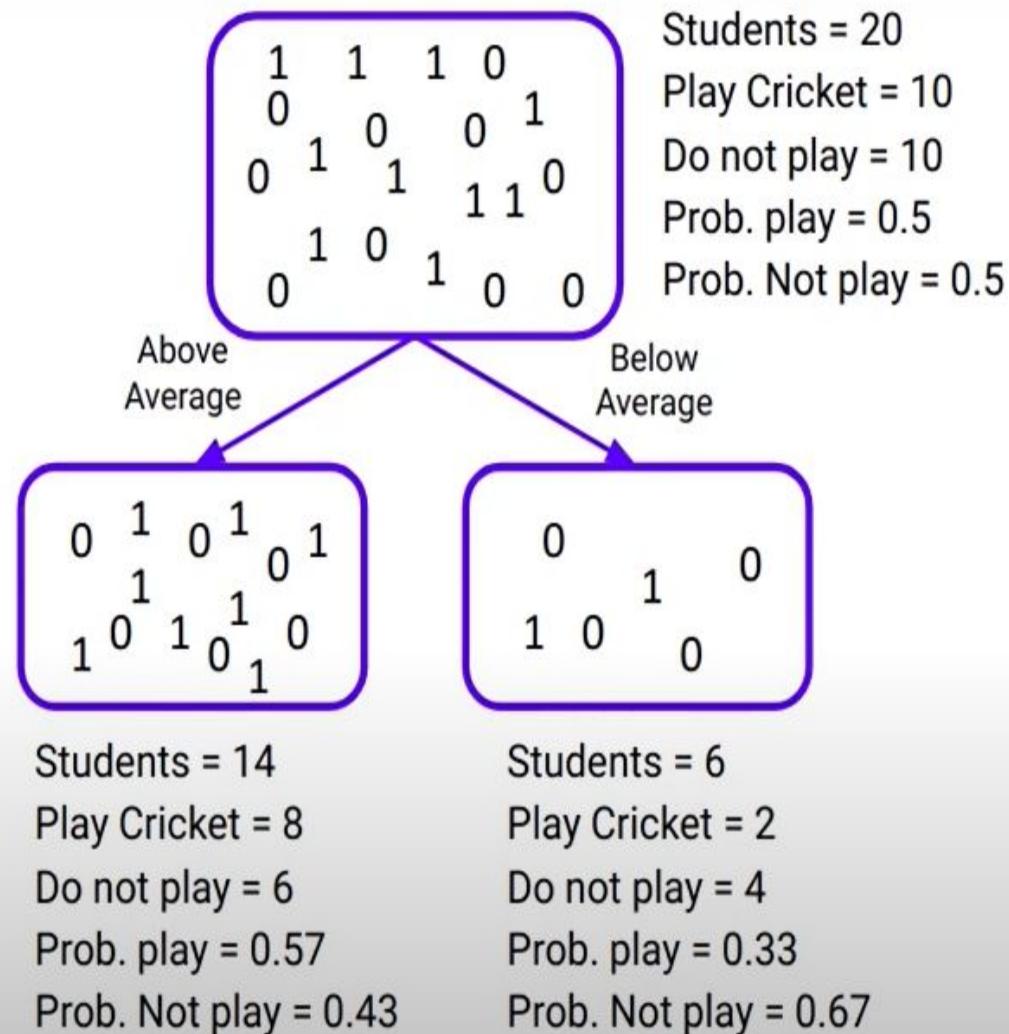
- Mean = $(2*1 + 4*0) / 6 = 0.33$

- Variance =

$$[2*(1-0.33)^2 + 4*(0-0.33)^2] / 6 = 0.222$$

- Variance: Performance in Class:

$$(14/20)*0.245 + (6/20)*0.222 = 0.238$$



Split	Variance
Performance in Class	0.238
Class	0.16

Entropy

- Is a measure of disorder in a system
- It is used to measure the impurity or randomness or uncertainty of a dataset

Entropy

- When a target feature contains more than one type of element (balls of different colors in a box), it is useful to sum up the entropies of each possible target value and weigh it by the probability of getting these values assuming a random draw
- This finally leads us to the formal definition of Shannon's entropy which serves as the baseline for the information gain calculation:

$$\text{Entropy}(x) = - \sum (P(x=k) * \log_2(P(x=k)))$$

- Where $P(x=k)$ is the probability that a target feature takes a specific value, k
- Or Entropy (S)= $P_+ (-\log_2 P_+) + P_- (-\log_2 P_-)$ where $P+$ and $P-$ are fraction of positive and negative examples in Sample S

Entropy

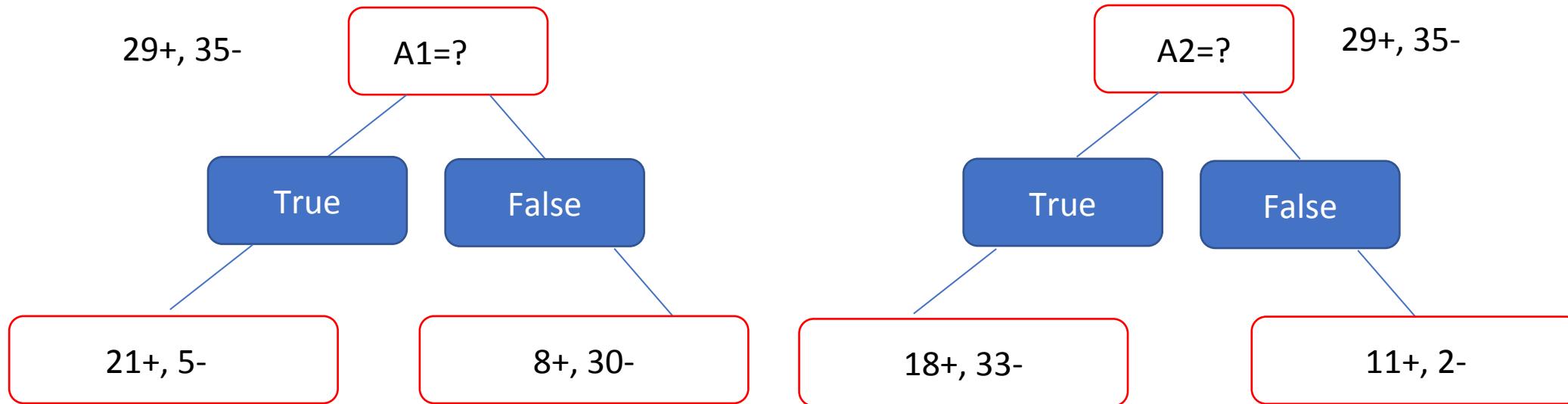
- Logarithm of fractions gives a negative value and hence a ‘-‘ sign is used in entropy formula to negate these negative values
- The maximum value for entropy depends on the number of classes

Information Gain

- Measures how well a given attribute separates the training examples according to their target classification
- This measure is used to select among the candidate attributes at each step while growing the tree.
- Gain is a measure of a reduction of uncertainty
- 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
- A decision tree algorithm always tries to maximize the value of information gain,
- A node/attribute having the highest information gain is split first
- $\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$
- ID3 algorithm uses information gain for constructing the decision tree

Information Gain

- $\text{Gain}(S, A)$: expected reduction in entropy due to partitioning S on attribute A



For previous example:

- Entropy([29+,35-]) = $-29/64 \log_2 29/64 - 35/64 \log_2 35/64 = 0.99$
- Entropy([21+,5-]) = 0.71 Entropy([18+,33-]) = 0.94
- Entropy([8+,30-]) = 0.71 Entropy([11+,2-]) = 0.94
- Gain(S,A1)= Entropy(S) – $26/64 * \text{Entropy}([21+,5-]) - 38/64 * \text{Entropy}([8+,30-]) = 0.27$
- Gain(S,A2)= Entropy(S) – $51/64 * \text{Entropy}([18+,33-]) - 13/64 * \text{Entropy}([11+,2-]) = 0.12$
- So according this measure A1 gets selected for splitting

Example

name	age	apple_pie?	potoato_salad?	sushi?	midwest?
Jeff	32	0	1	1	1
Pete	25	1	1	0	1
Anne	33	1	1	0	1
Natalie	26	0	0	1	0
Stella	30	1	1	1	1
Rob	25	1	0	0	1
Joe	42	1	1	0	1
Jim	38	1	1	0	1
Lisa	36	1	1	0	0
Sarah	29	1	0	1	0
David	35	1	0	0	1
Eric	28	1	1	1	0
Mike	20	0	1	0	1
Karen	38	1	0	0	1
Megan	31	0	0	1	0

The goal of the quiz will be to guess if the quiz taker is from one of America's midwest states.

Entropy

- We then compute the probability of that value occurring in the data.
 - For the case of (1), the probability is **10/15**.
 - For the case of (0), the probability is **5/15**.
 - We take the probability of each case and multiply it by the logarithm base 2 of the probability.
 - For the case of(1), we get **10/15*log2(10/15)**.
 - For the case of (0), we get **5/15*log2(5/15)**.
 - Next, we take our product from each case above and sum it together. For this example, **-{10/15*log2(10/15) + 5/15*log2(5/15)}= .918278**

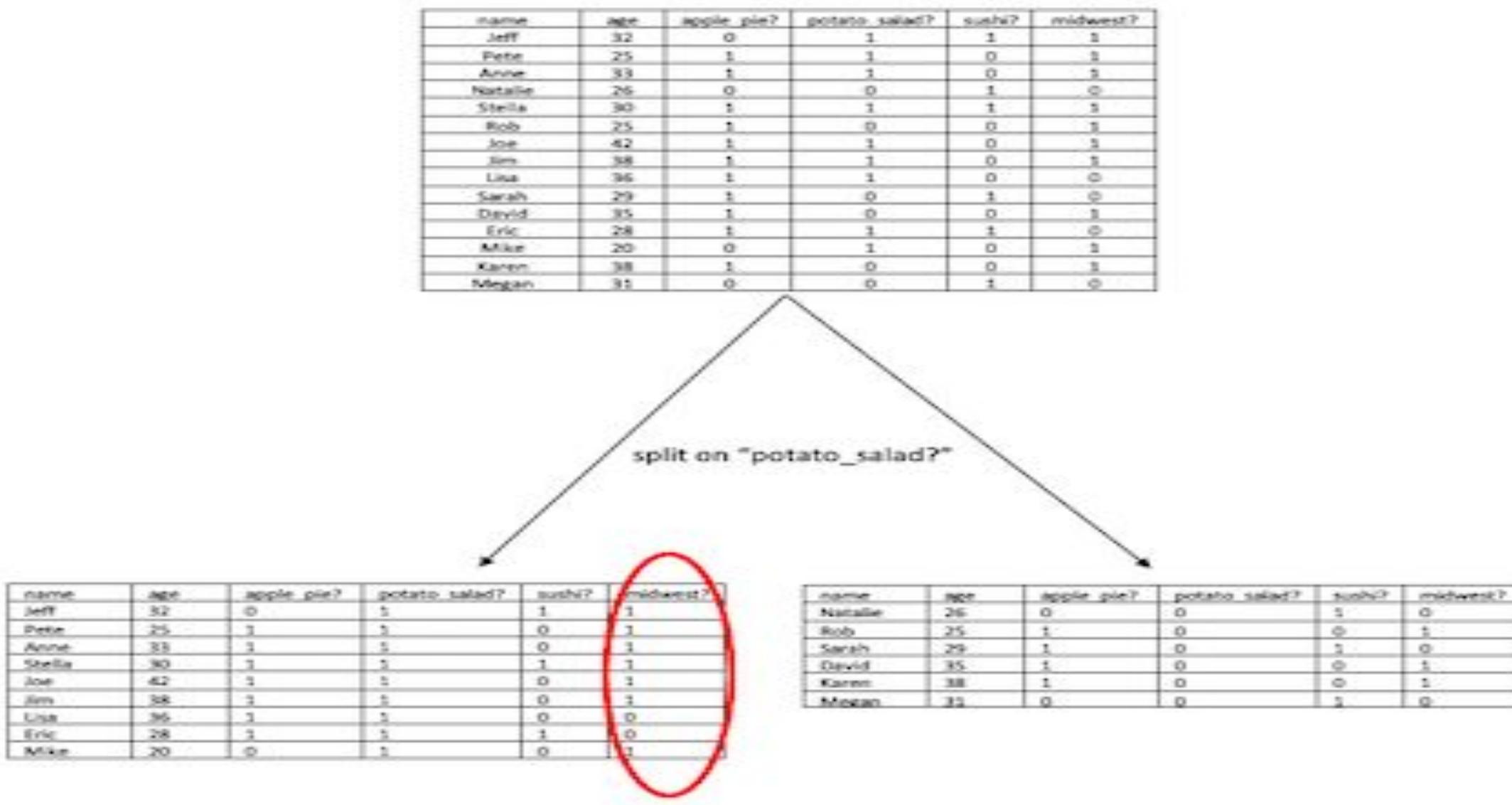
midwest?
1
1
1
0
1
1
1
1
0
0
1
0
1
1
0

$$-\sum_{i=1}^c P(x_i) \log_b P(x_i)$$

Goal in building the Decision tree

- Our goal is to find the best variable(s)/column(s) to split on when building a decision tree.
- Eventually, we want to keep splitting the variables/columns until our mixed target column is no longer mixed.
- For instance, let's look at the entropy of the “midwest?” column after we have split our dataset on the “potato_salad?” column.

Splitting the dataset on the “potato salad”



Calculating Information gain

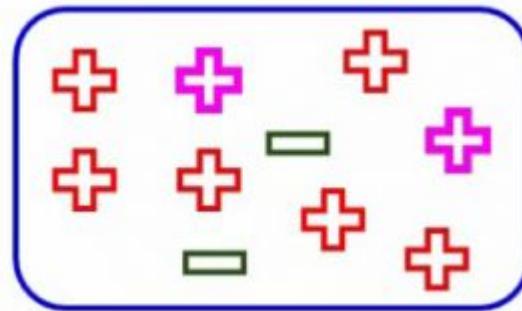
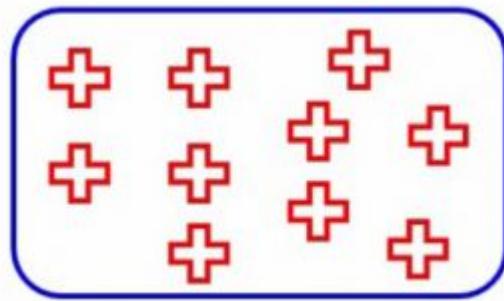
- for each unique value (v) in the feature considered, we compute the number of rows which takes on the value (v), and divide it by the total number of rows.
 - For the left split(split on 1 for “potato_salad?”) we get $-7/9 * \log_2 (7/9)$
 - For the right side of the split (split on 0 for “potato_salad?”) we get $-2/9 * \log_2 (2/9)$
- Weighted entropy of split= $9/15 * \{-7/9 * \log_2 (7/9)\} + 6/15 * \{-2/9 * \log_2 (2/9)\}$
- We then subtract from the overall entropy to get information gain
 - $.918278 - [9/15 * \{-7/9 * \log_2 (7/9)\} + 6/15 * \{-2/9 * \log_2 (2/9)\}]$
- This process is repeated for every column

Another (Splitting Rule):

- Gini Index/Gini Impurity
- Measure of node impurity
- A Decision tree algorithm for selecting the best split
- Used in CART

Gini

- If we pick two points from a population at random what is the probability of getting both points from same class (the pink ones highlighted here)



- completely pure node-

- probability that randomly picked point will belong to the same class-1
- Gini ranges from zero to one, as it is a probability
 - The higher this value, the more will be the purity of the nodes.
 - a lesser value means lesser pure nodes.
- It is the sum of squared probabilities of each class

$$Gini = (p_1^2 + p_2^2 + p_3^2 + \dots + p_n^2)$$

Gini Index/Gini Impurity

- Measure of node impurity
- Gini index is a measure of impurity used while creating a decision tree

$$\text{Gini Impurity} = 1 - \text{Gini}$$

- CART algorithm uses the Gini index to create binary splits
- $\text{GINI}_{\text{node}}(\text{Node}) = 1 - \sum_{c \in \text{classes}} [p(c)]^2$
- $\text{GINI}_{\text{split}}(A) = \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{GINI}(N_v)$

Gini Index/Gini Impurity

- Gini index or Gini impurity measures the degree or probability of a particular variable being wrongly classified when it is randomly chosen.
- The degree of Gini index varies between 0 and 1, where,
 - 0 denotes that all elements belong to a certain class or if there exists only one class, and
 - 1 denotes that the elements are randomly distributed across various classes.
 - 0.5 denotes equally distributed elements into some classes.

Properties of Gini impurity

- We decide the best split based on the Gini impurity

$$\text{Gini Impurity} = 1 - \text{Gini}$$

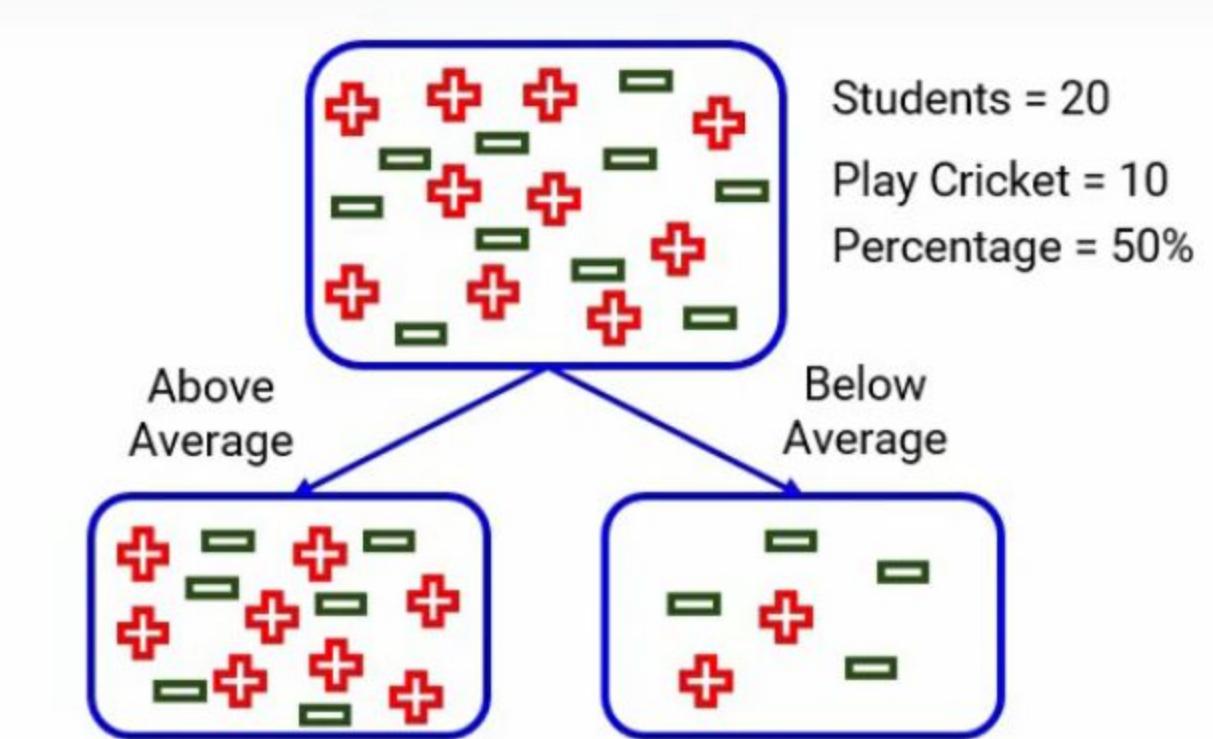
- Gini denotes the purity and hence Gini impurity tells us about the impurity of nodes
- Lower the Gini impurity we can safely infer the purity will be more and hence a higher chance of the homogeneity of the nodes.

Properties of Gini impurity

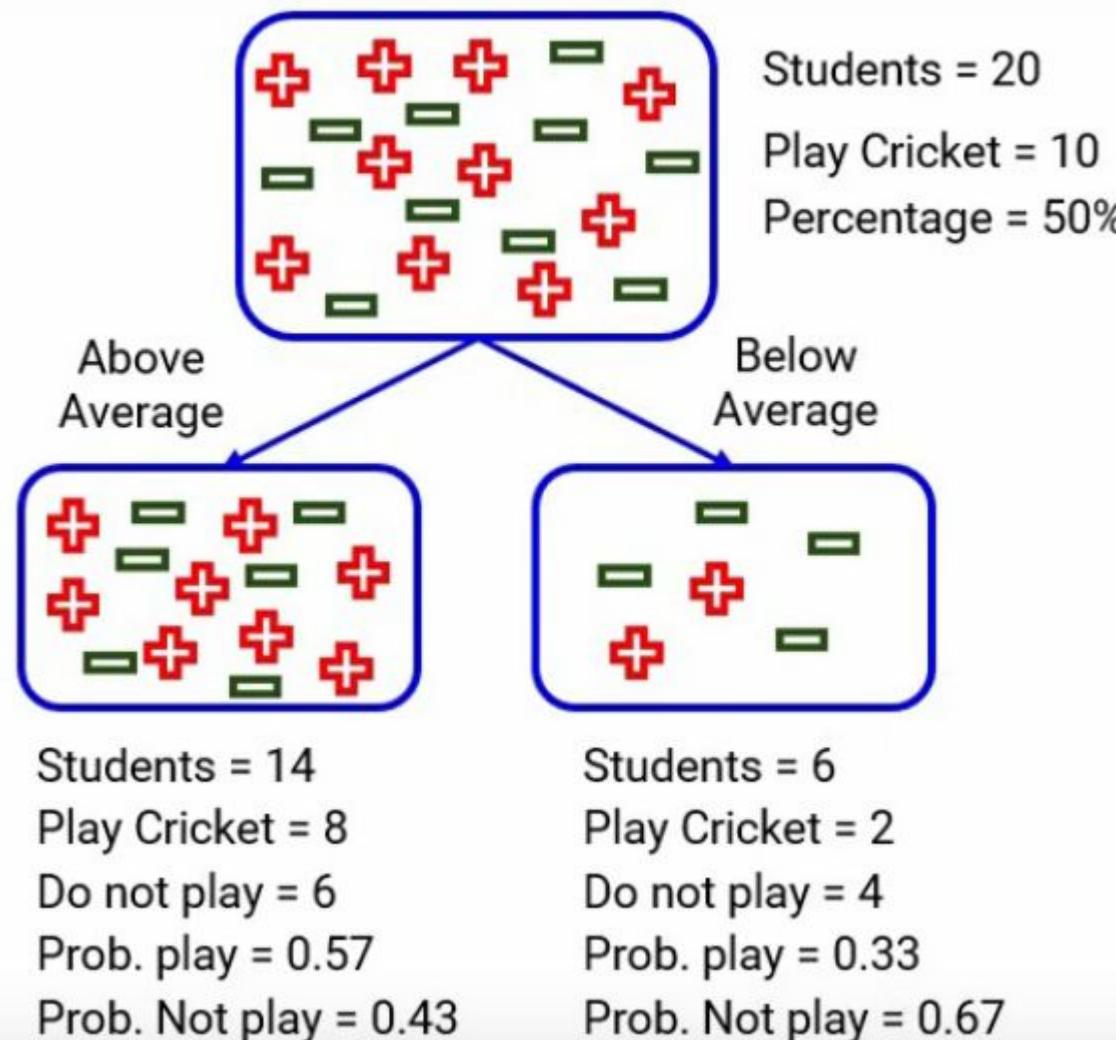
- Gini works only in those scenarios where we have **categorical** targets.
- It does not work with continuous targets.
- For example, if you want to predict the house price or the number of bikes that have been rented, Gini is not the right algorithm.

Steps to calculate Gini Impurity

- 1. Calculate the Gini impurity for sub-nodes
- 2. Calculate the Gini impurity of the split using the weighted impurity of both sub-nodes of that split.
 - Here the weight is decided by the number of observations of samples in both the nodes.



Split on Performance in Class



Gini Impurity: sub-node Above Average:
 $1 - [(0.57)*(0.57) + (0.43)*(0.43)] = 0.49$

Gini Impurity: sub-node Below Average:
 $1 - [(0.33)*(0.33) + (0.67)*(0.67)] = 0.44$

Weighted Gini Impurity: Performance in Class:
 $(14/20)*0.49 + (6/20)*0.44 = 0.475$

Split on Class

Gini Impurity: sub-node Class IX:

$$1 - [(0.8) * (0.8) + (0.2) * (0.2)] = 0.32$$

Gini Impurity: sub-node Class X:

$$1 - [(0.2) * (0.2) + (0.8) * (0.8)] = 0.32$$

Weighted Gini Impurity: Class:

$$(10/20) * 0.32 + (10/20) * 0.32 = 0.32$$

Split	Weighted Gini Impurity
Performance in Class	0.475
Class	0.32

Example of Gini Index

Past Trend	Open Interest	Trading Volume	Return
Positive	Low	High	Up
Negative	High	Low	Down
Positive	Low	High	Up
Positive	High	High	Up
Negative	Low	High	Down
Positive	Low	Low	Down
Negative	High	High	Down
Negative	Low	High	Down
Positive	Low	Low	Down
Positive	High	High	Up

Calculating the Gini Index for Past Trend

- $P(\text{Past Trend}=\text{Positive})$: 6/10
- $P(\text{Past Trend}=\text{Negative})$: 4/10
- If (Past Trend = Positive & Return = Up), probability = 4/6
- If (Past Trend = Positive & Return = Down), probability = 2/6
- Gini index = $1 - ((4/6)^2 + (2/6)^2) = 0.45$
- If (Past Trend = Negative & Return = Up), probability = 0
- If (Past Trend = Negative & Return = Down), probability = 4/4
- Gini index = $1 - ((0)^2 + (4/4)^2) = 0$
- Weighted sum of the Gini Indices can be calculated as follows:
- **Gini Index for Past Trend = $(6/10)0.45 + (4/10)0 = 0.27$**

Calculation of Gini Index for Open Interest

- $P(\text{Open Interest}=\text{High})$: 4/10
- $P(\text{Open Interest}=\text{Low})$: 6/10
- If (Open Interest = High & Return = Up), probability = 2/4
- If (Open Interest = High & Return = Down), probability = 2/4
- Gini index = $1 - ((2/4)^2 + (2/4)^2) = 0.5$
- If (Open Interest = Low & Return = Up), probability = 2/6
- If (Open Interest = Low & Return = Down), probability = 4/6
- Gini index = $1 - ((2/6)^2 + (4/6)^2) = 0.45$
- Weighted sum of the Gini Indices can be calculated as follows:
- **Gini Index for Open Interest = $(4/10)0.5 + (6/10)0.45 = 0.47$**

Calculation of Gini Index for Trading Volume

- $P(\text{Trading Volume}=\text{High})$: 7/10
- $P(\text{Trading Volume}=\text{Low})$: 3/10
- If (Trading Volume = High & Return = Up), probability = 4/7
- If (Trading Volume = High & Return = Down), probability = 3/7
- Gini index = $1 - ((4/7)^2 + (3/7)^2) = 0.49$
- If (Trading Volume = Low & Return = Up), probability = 0
- If (Trading Volume = Low & Return = Down), probability = 3/3
- Gini index = $1 - ((0)^2 + (1)^2) = 0$
- Weighted sum of the Gini Indices can be calculated as follows:
- **Gini Index for Trading Volume = $(7/10)0.49 + (3/10)0 = 0.34$**

Gini Index attributes or features

Attributes/Features	Gini Index
Past Trend	0.27
Open Interest	0.47
Trading Volume	0.34

From the above table, we observe that 'Past Trend' has the lowest Gini Index and hence it will be chosen as the root node for how decision tree works.

We will repeat the same procedure to determine the sub-nodes or branches of the decision tree.

Gini Index for the ‘Positive’ branch of Past Trend

Past Trend	Open Interest	Trading Volume	Return
Positive	Low	High	Up
Positive	Low	High	Up
Positive	High	High	Up
Positive	Low	Low	Down
Positive	Low	Low	Down
Positive	High	High	Up

Calculation of Gini Index of Open Interest for Positive Past Trend

- $P(\text{Open Interest}=\text{High})$: 2/6
- $P(\text{Open Interest}=\text{Low})$: 4/6
- If (Open Interest = High & Return = Up), probability = 2/2
- If (Open Interest = High & Return = Down), probability = 0
- Gini index = $1 - (\text{sq}(2/2) + \text{sq}(0)) = 0$
- If (Open Interest = Low & Return = Up), probability = 2/4
- If (Open Interest = Low & Return = Down), probability = 2/4
- Gini index = $1 - (\text{sq}(0) + \text{sq}(2/4)) = 0.50$
- Weighted sum of the Gini Indices can be calculated as follows:
- **Gini Index for Open Interest = $(2/6)0 + (4/6)0.50 = 0.33$**

Calculation of Gini Index for Trading Volume for Positive Past Trend

- $P(\text{Trading Volume}=\text{High})$: 4/6
- $P(\text{Trading Volume}=\text{Low})$: 2/6
- If (Trading Volume = High & Return = Up), probability = 4/4
- If (Trading Volume = High & Return = Down), probability = 0
- Gini index = $1 - (\text{sq}(4/4) + \text{sq}(0)) = 0$
- If (Trading Volume = Low & Return = Up), probability = 0
- If (Trading Volume = Low & Return = Down), probability = 2/2
- Gini index = $1 - (\text{sq}(0) + \text{sq}(2/2)) = 0$
- Weighted sum of the Gini Indices can be calculated as follows:
- **Gini Index for Trading Volume = $(4/6)0 + (2/6)0 = 0$**

Gini Index attributes or features

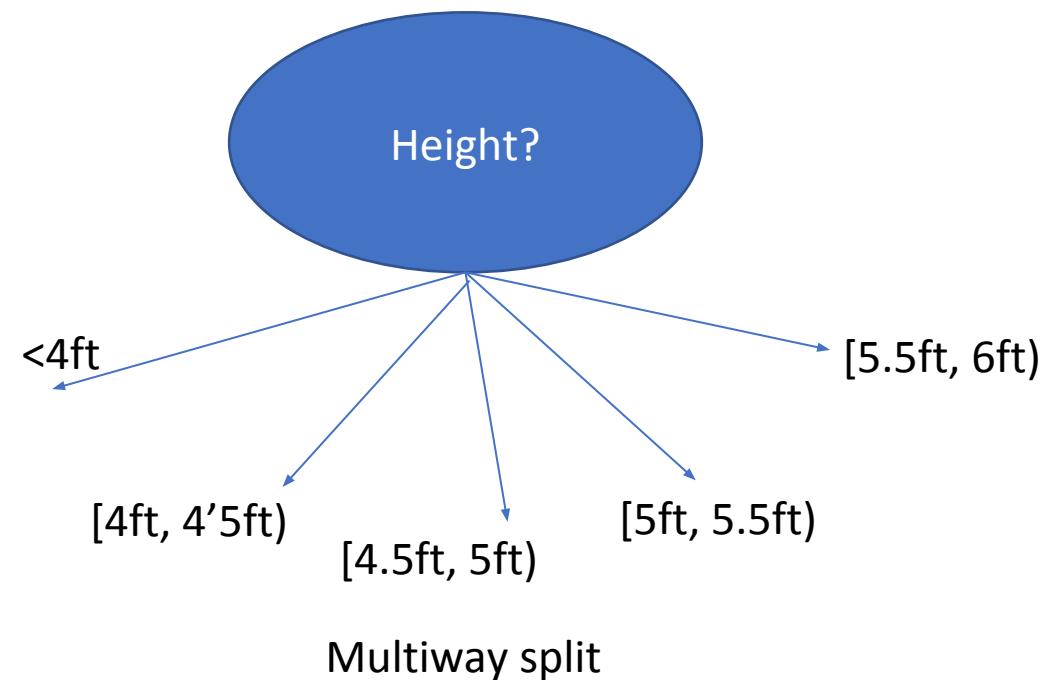
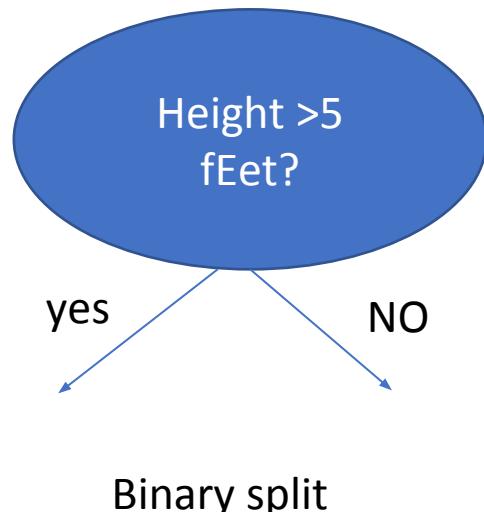
Attributes/Features	Gini Index
Open Interest	0.33
Trading Volume	0

We will split the node further using the ‘Trading Volume’ feature, as it has the minimum Gini index.

- Gini Index, unlike information gain, isn't computationally intensive as it doesn't involve the logarithm function used to calculate entropy in information gain.
- This is why Gini Index is preferred over Information gain.

Splitting based on continuous attribute

- We discussed decision tree if training example contains an attribute having 2 or 3 or 4 values i.e. if attributes are nominal values
- What if the training examples have real valued examples



Continuous Attribute - Binary Split

- For continuous attributes
 - Partition the continuous value of attribute A into a discrete set of intervals
 - Create a new Boolean attribute A_c , looking for a threshold c
 - $$A_c = \begin{cases} \text{true} & \text{if } A_c < c \\ \text{False} & \text{otherwise} \end{cases}$$
 - How to choose c? (what is the value on which you will split, if height then 4? Or 4.5? Or 5? Etc.)
 - Consider all possible splits and find the best cut (where the information gain is maximum)
 - Computationally intensive but one can do it intelligently

Dealing with continuous-values attributes

- Example: temperature in the playTennis
- Sort the examples according to Temperature
- Temperature 40 48 60 72 80 90
- PlayTennis No No Yes Yes Yes No
- Steps to decide the value of temperature to split the node
 - 1. Determine the candidate thresholds by averaging consecutive values where there is a change in classification: $(48+60)/2 = 54$ and $(80+90)/2 = 85$
 - 2. Evaluate information gain of candidate thresholds (attributes) $\text{temperature} > 54$ and $\text{Temperature} > 85$
 - 3. Select threshold based on the information gain

- Entropy of whole dataset (S)
- $S = [3+, 3-]$ $\text{Entropy}(S) = -\frac{3}{6} \log_2 \frac{3}{6} - \frac{3}{6} \log_2 \frac{3}{6} = 1.0$
- $S_{T < 54} = [0+, 2-]$ $\text{Entropy}(S_{T < 54}) = 0.0$
- $S_{T > 54} = [3+, 1-]$ $\text{Entropy}(S_{T > 54}) = -\frac{3}{4} \log_2 \frac{3}{4} - \frac{1}{4} \log_2 \frac{1}{4} = 0.8113$
- $\text{Gain}(S, T > 54) = \text{Entropy}(S) - \frac{2}{6} \times \text{Entropy}(S_{T < 54}) - \frac{4}{6} \times \text{Entropy}(S_{T > 54})$
 $= 0.4591$

- Entropy of whole dataset (S)
- $S = [3+, 3-]$ $\text{Entropy}(S) = -\frac{3}{6} \log_2 \frac{3}{6} - \frac{3}{6} \log_2 \frac{3}{6} = 1.0$
- $S_{T < 84} = [3+, 2-]$ $\text{Entropy}(S_{T < 84}) = -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} = 0.971$
- $S_{T > 84} = [0+, 1-]$ $\text{Entropy}(S_{T > 84}) = 0.0$
- $\text{Gain}(S, T > 84) = \text{Entropy}(S) - \frac{5}{6} \times \text{Entropy}(S_{T < 84}) - \frac{1}{6} \times \text{Entropy}(S_{T > 84})$
 $= 0.1908$

- $\text{Gain}(S, T>54) = 0.4591$
- $\text{Gain}(S, T>84) = 0.1908$

Decision Tree Splitting: Chi-Square

- Another method of splitting nodes in a decision tree for datasets having categorical target values.
- It can make two or more than two splits.
- It works on the statistical significance of differences between the parent node and child nodes.
- It is measured as the sum of squared standardized differences between observed and expected frequencies of target variable for each node
- Chi-Square value is:

$$Chi-Square = \sqrt{\frac{(Actual - Expected)^2}{Expected}}$$

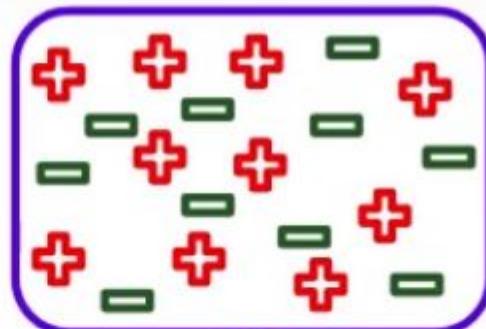
Example

Split on Performance in Class

Students = 14

Expected Play = 7
Play Cricket = 8

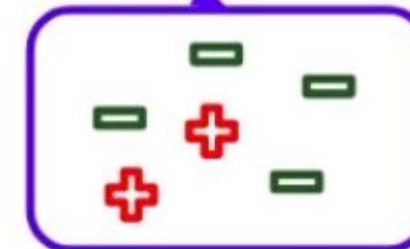
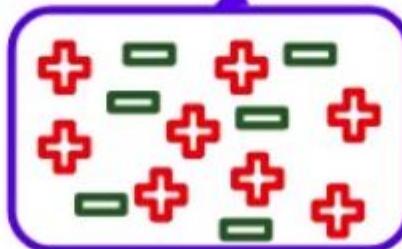
Expected Not Play = 7
Not Play Cricket = 6



Students = 20
Play Cricket = 10
Do not play = 10
Percentage play = 50%
Percentage not play = 50%

Above Average

Below Average



- There is a total of 20 students and out of those 10 play cricket and 10 do not.
- So, of course, the percent of students who do play cricket will be 50%.
- Now if we consider the “Above average” node here,
 - there are 14 students in it, as the percentage of students who play cricket is 50% in the parent node as we discussed,
 - the expected number of students who play cricket will of course be 7 and if you look at the actual value it is 8.
 - So expected value-7 and actual value-8
 - Similarly, the expected number of students who do not play cricket will be 7 and the actual value turns out to be 6.
- Similarly for the below-average node expected to play and not play will be 3. Whereas the actual values are- 2 students play cricket and 4 do not.

Node	Actual Play	Actual Not Play	Expected Play	Expected Not Play
Above Average	8	6	7	7
Below Average	2	4	3	3

Node	Actual Play	Actual Not Play	Expected Play	Expected Not Play	Deviation Play	Deviation Not Play
Above Average	8	6	7	7	1	-1
Below Average	2	4	3	3	-1	1

Node	Actual Play	Actual Not Play	Expected Play	Expected Not Play	Deviation Play	Deviation Not Play	Chi-Square (Play)	Chi-Square (Not Play)
Above Average	8	6	7	7	1	-1	0.38	0.38
Below Average	2	4	3	3	-1	1	0.58	0.58

Chi-square for the split in “performance in class” will be the sum of all these chi-square values:

$$\text{Total} = 0.38 + 0.38 + 0.58 + 0.58 = 1.92$$

Split on class

Node	Actual Play	Actual Not Play	Expected Play	Expected Not Play	Deviation Play	Deviation Not Play	Chi-Square (Play)	Chi-Square (Not Play)
IX	8	2	5	5	3	-3	1.34	1.34
X	2	8	5	5	-3	3	1.34	1.34

chi-square value for the split on class which will be the sum of all chi-square values:

$$\text{Total} = 1.34 + 1.34 + 1.34 + 1.34 = 5.36$$

Split	Chi-Square
Performance in Class	1.92
Class	5.36

- what will be the chi-square value if the actual and expected values are the same?
 - It will be **zero** because both the actual and expected are the same and the difference will be zero.
 - In this case, distribution of the child node is the same as there is the parent node and hence we are not improving the purity of the nodes.
- Higher the chi-square value more will be the purity of the nodes after a split.

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

Importing the libraries

- `import numpy as np`
- `import matplotlib.pyplot as plt`
- `import pandas as pd`

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
```

```
x = dataset.iloc[:, :-1].values
```

```
y = dataset.iloc[:, -1].values
```

	Age	EstimatedSalary	Purchased
	19	19000	0
	35	20000	0
	26	43000	0
	27	57000	0
	19	76000	0
	27	58000	0
	27	84000	0
	32	150000	1
	25	33000	0

Total- 1000 entries

Splitting the dataset into the Training set and Test set

- `from sklearn.model_selection import train_test_split`
- `x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)`

Feature Scaling

- `from sklearn.preprocessing import StandardScaler`
- `r`
- `sc = StandardScaler()`
- `X_train = sc.fit_transform(X_train)`
- `X_test = sc.transform(X_test)`

Training the Decision Tree Classification model on the Training set

- `from sklearn.tree import DecisionTreeClassifier`
- `classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)`
- `classifier.fit(X_train, y_train)`

Predicting a new result

- `print(classifier.predict(sc.transform([[30, 8700
0]])))`
- Output
 - [0]

Predicting the Test set results

- `y_pred = classifier.predict(X_test)`
- `print(np.concatenate((y_pred.reshape(len(y_pred)), 1), y_test.reshape(len(y_test), 1)), 1))`

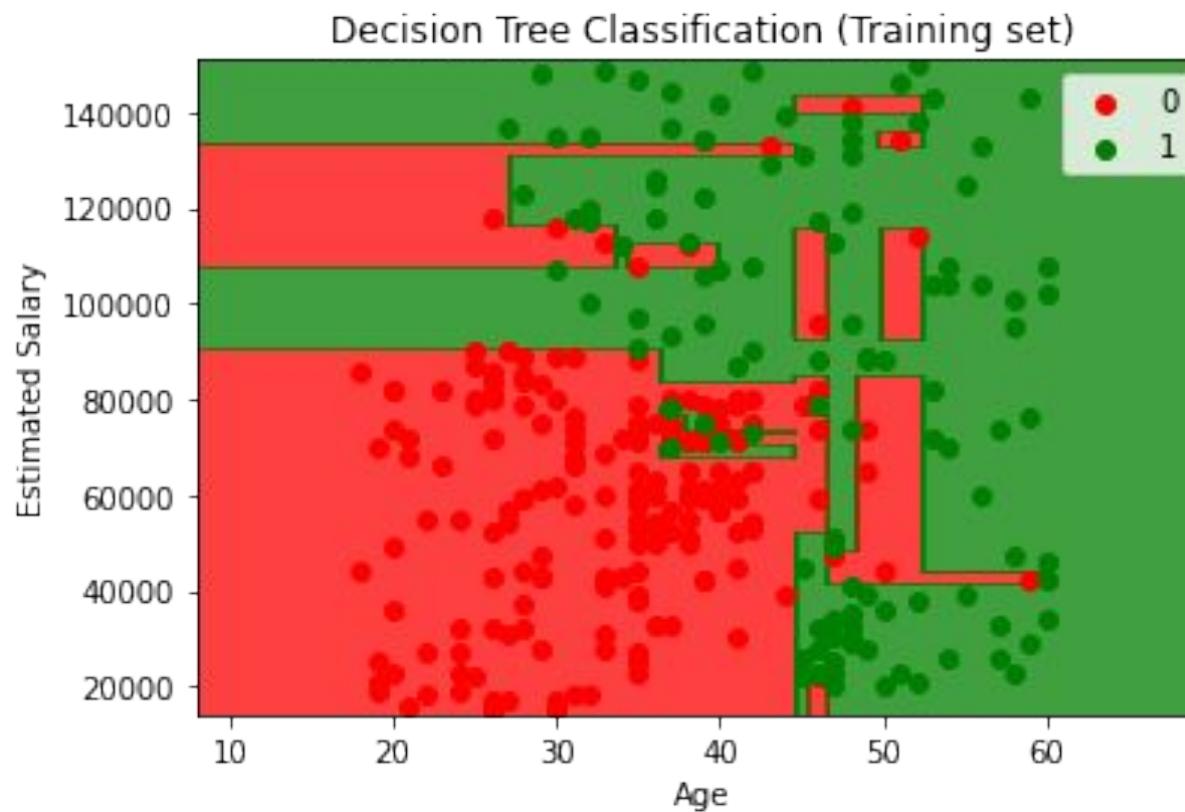
Making the Confusion Matrix

- `from sklearn.metrics import confusion_matrix, accuracy_score`
- `cm = confusion_matrix(y_test, y_pred)`
- `print(cm)`
- `accuracy_score(y_test, y_pred)`
- **Output**
[[62 6]
 [3 29]
0.91

Visualising the Training set results

```
• from matplotlib.colors import ListedColormap  
• X_set, y_set = sc.inverse_transform(X_train), y_train  
• X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),  
•                      np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))  
• plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),  
•               alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
• plt.xlim(X1.min(), X1.max())  
• plt.ylim(X2.min(), X2.max())  
• for i, j in enumerate(np.unique(y_set)):  
•     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)  
• plt.title('Decision Tree Classification (Training set)')  
• plt.xlabel('Age')  
• plt.ylabel('Estimated Salary')  
• plt.legend()  
• plt.show()
```

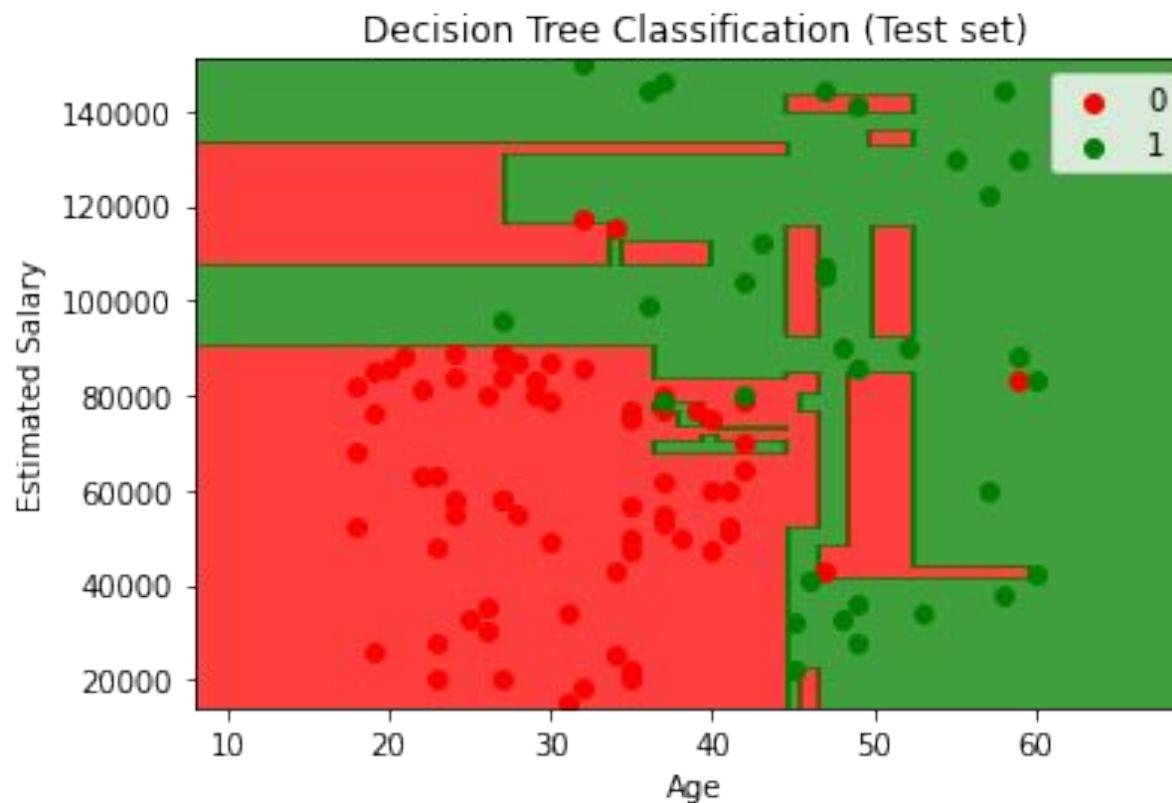
Visualising the Training set results



Visualising the Test set results

```
• from matplotlib.colors import ListedColormap  
• X_set, y_set = sc.inverse_transform(X_test), y_test  
• X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),  
•                      np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))  
• plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),  
•               alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
• plt.xlim(X1.min(), X1.max())  
• plt.ylim(X2.min(), X2.max())  
• for i, j in enumerate(np.unique(y_set)):  
•     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)  
• plt.title('Decision Tree Classification (Test set)')  
• plt.xlabel('Age')  
• plt.ylabel('Estimated Salary')  
• plt.legend()  
• plt.show()
```

Visualising the Test set results



Occam's razor

- Occam's Razor is an approach to problem-solving and is commonly invoked to mean that if all else is equal, we should prefer the simpler solutions.
- **Occam's Razor:** If all else is equal, the simplest solution is correct.
- It is not a rule, more of a heuristic for problem-solving, and is commonly invoked in science to prefer simpler hypotheses that make fewer assumptions over more complex hypotheses that make more assumptions.

Occam's Razor for Model Selection

- model selection is based on its expected performance, e.g.
 - highest accuracy or lowest prediction error.
- Another important consideration is
 - to choose simpler models over complex models.
- Simpler models are typically defined as
 - that make fewer assumptions or have fewer elements, most commonly characterized as fewer coefficients (e.g. rules, layers, weights, etc.).
 - The rationale for choosing simpler models is tied back to Occam's Razor.

Occam's razor

- Suggests that in machine learning, we should prefer simpler models with fewer coefficients over complex models like ensembles.
- It is a heuristic that suggests
 - Choosing simpler machine learning models as they are expected to generalize better.
 - more complex hypotheses makes more assumptions that, in turn, will make them too narrow and not generalize well.
- It suggests complex models like ensembles will overfit the training dataset and perform poorly on new data.
- <https://machinelearningmastery.com/ensemble-learning-and-occams-razor/>

- But empirical results show a continued reduction in generalization error as the complexity of an ensemble learning model is incrementally increased.
 - *It has been empirically observed that certain ensemble techniques often do not overfit the model, even when the ensemble contains thousands of classifiers.*
- These findings are at odds with the Occam's razor principle taken at face value.
- How can this inconsistency be reconciled?

- Occam's razor is a heuristic that suggests
 - choosing simpler machine learning models as they are expected to generalize better
- The heuristic can be divided into two razors
 - one of which is true and remains a useful tool and
 - the other that is false and should be abandoned
- Specific case of how the second razor fails
 - Ensemble learning algorithms like boosting
- Thus, it shows added complexity can result in lower generalization error

Occam's Two Razors for Machine Learning

- The conflict between the expectation of simpler models generalizing better in theory and complex models like ensembles generalizing better in practice was mostly ignored as an inconvenient empirical finding for a long time.
- In the late 1990s, the problem was specifically studied by Pedro Domingos and published in the award-winning 1996 paper titled “Occam’s Two Razors: The Sharp and the Blunt,” and follow-up 1999 journal article “The Role of Occam’s Razor in Knowledge Discovery.”

- In the work, Domingos defines the problem as two specific commonly asserted implications of Occam's Razor in applied machine learning, which he refers to as "*Occam's Two Razors*" in machine learning, they are (taken from the paper):
- **First razor:** Given two models with the same generalization error, the simpler one should be preferred because simplicity is desirable in itself.
- **Second razor:** Given two models with the same training-set error, the simpler one should be preferred because it is likely to have lower generalization error.
- Domingos then enumerates a vast number of examples for and against each razor from both theory and empirical studies in machine learning.

- The **first razor** suggests if two models have the same expected performance on data not seen during training, we should prefer the simpler model.
 - Domingos highlights that this razor holds and provides a good heuristic on machine learning projects.
- The **second razor** suggests that if two models have the same performance on a training dataset, then the simpler model should be chosen because it is expected to generalize better when used to make predictions on new data.
 - This seems sensible on the surface.
 - It is the argument behind not adopting ensemble algorithms in a machine learning project because they are very complex compared to other models and expected to not generalize.
 - It turns out that **this razor cannot be supported** by the evidence from the machine learning literature.

Occam's Razor and Ensemble Learning

- in practice, we would not choose a machine learning model based on its performance on the training dataset alone.
- We intuitively, or perhaps after a lot of experience, tacitly expect the estimate of performance on a training set to be a poor estimate of performance on a holdout dataset.
- We have this expectation because the model can overfit the training dataset.
- Yet, less intuitively, overfitting the training dataset can lead to better performance on a holdout test set. This has been observed many times in practice in systematic studies.

- A common situation involves plotting the performance of a model on the training dataset and a holdout test dataset each iteration of learning for the model, such as training epochs or iterations for models that support incremental learning.
- If learning on the training dataset is set up to continue for a large number of training iterations and the curves observed, it can often be seen that performance on the training dataset will fall to zero error. This is to be expected as we might think that the model will overfit the training dataset given enough resources and time to train. Yet the performance on the test set will continue to improve, even while the performance on the training set remains fixed at zero error.
 - *occasionally, the generalization error would continue to improve long after the training error had reached zero*

- This behavior can be observed with ensemble learning algorithms like boosting and bagging, where performance on the holdout dataset will continue to improve as additional model members are added to the ensemble.
 - *One very surprising finding is that performing more boosting iterations can reduce the error on new data long after the classification error of the combined classifier on the training data has dropped to zero.*

- That is, the model complexity is incrementally increased, which systematically decreases the error on unseen data, e.g. generalization error. The additional training cannot improve the performance on the training dataset; it has no possible improvement to make.
 - *Performing more boosting iterations without reducing training error does not explain the training data any better, and it certainly adds complexity to the combined classifier.*
- This finding directly contradicts the second razor and supports Domingos' argument about abandoning the second razor.
 - *The first one is largely uncontroversial, while the second one, taken literally, is false.*

- This problem has been studied and can generally be explained by the ensemble algorithms learning to be more confident in their predictions on the training dataset, which carry over to the hold out data.
 - *The contradiction can be resolved by considering the classifier's confidence in its predictions.*

- The first razor remains an important heuristic in applied machine learning.
- This is not the only way to choose models.
- We may choose a simpler model because it is easier to interpret, and this remains valid if model interpretability is a more important project requirement than predictive skill.
-

Generalization error

- Expected prediction error over on independent test data
- Aim
 - Minimize the generalization error as much as possible
- Generalization error estimate in decision tree is done with two ways
 - Optimistic error estimate/ resubstitution error
 - Pessimistic error estimate

Measures of Similarity and Dissimilarity, Cross-Validation

Measures of similarity and dissimilarity

Similarity measures

- To deal with continuous or numerical variables
 - Euclidean Distance
 - Manhattan Distance
 - Minkowski Distance
 - Cosine similarity
 - Jacard Similarity
 - Pearson Correlation Coefficient
- To deal with categorical variables
 - Hamming Distance

Formula for Euclidean Distance

$$d = ((p_1 - q_1)^2 + (p_2 - q_2)^2)^{1/2}$$

$$D_e = \left(\sum_{i=1}^n (p_i - q_i)^2 \right)^{1/2}$$

Where,

- n = number of dimensions
- pi, qi = data points

Manhattan Distance

- is the sum of absolute differences between points across all the dimensions

$$d = |p_1 - q_1| + |p_2 - q_2|$$

$$D_m = \sum_{i=1}^n |p_i - q_i|$$

Where,

- n = number of dimensions
- pi, qi = data points

Minkowski Distance

- Minkowski Distance is the generalized form of Euclidean and Manhattan Distance.

$$D = \left(\sum_{i=1}^n |p_i - q_i|^p \right)^{1/p}$$

Cosine Similarity

- Measures the cosine of the angle between two vectors
- $\text{Similarity}(x,y) = x \cdot y / \|x\| \|y\|$
- Range: [0, 1] (or [-1, 1] in some variations)
- Usage
 - Text analysis, document similarity, and recommendation systems

Jaccard Similarity

- Measures similarity between two sets by comparing their intersection to their union
- Formula: $J(A,B) = |A \cap B| / |A \cup B|$
- Usage: Set-based data such as binary attributes or text data

Pearson Correlation Coefficient

- Measures linear correlation between two variables
- $r = [\sum(x_i - \bar{x})(y_i - \bar{y})] / \sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}$
- Range: [-1, 1]
- Usage: Measures relationships in regression analysis

Hamming Distance

- measures the similarity between two strings of the same length
- is the number of positions at which the corresponding characters are different
- Usage
 - Useful for binary data and text data like DNA sequences

Dissimilarity Measures

- Euclidean Distance
- Manhattan Distance
- Mahalanobis Distance
- Canberra Distance
- Chebyshev Distance (Maximum Distance)
- Edit Distance (Levenshtein Distance)

Dissimilarity Measures

- Euclidean Distance (Same as similarity but interpreted as dissimilarity)
- Manhattan Distance (Same as similarity but interpreted as dissimilarity)

Mahalanobis Distance

- Euclidean Distance treats each dimension (feature) as independent
 - It assumes the same scale and importance for each dimension
- When dimensions are correlated, Euclidean distance can be misleading because it ignores these correlations
- Mahalanobis Distance takes into account the **covariance** between dimensions
 - $d(x,y)=\sqrt{(x-y)^T S^{-1} (x-y)}$
 - where S is the covariance matrix
- Usage
 - Useful when data dimensions are correlated

Canberra Distance

- $d(x,y)=\sum_{i=1}^n |x_i| - |y_i| / |x_i+y_i|$
- Sensitive to small changes, especially for small values

Chebyshev Distance (Maximum Distance)

- gives the largest difference among all dimensions and ignores the others
 - $d(x,y)=\max |x_i-y_i|$
- can be useful when the worst-case difference is more important than cumulative differences
- Usage
 - Chessboard-like movement scenarios, max-min optimization problems

Edit Distance (Levenshtein Distance)

- Measures the minimum number of operations (insertions, deletions, substitutions) to transform one string into another
- Usage
 - Text processing and DNA sequence alignment

Applicability of Metrics

- Continuous Data: Euclidean, Manhattan, Minkowski
- Categorical Data: Hamming, Jaccard
- High-Dimensional Data: Cosine similarity

Cross-validation

- is a resampling technique
 - used to evaluate the performance and generalizability of a machine learning model
- It involves
 - partitioning the dataset into multiple subsets (folds) and using different subsets for training and testing the model
- The goal is
 - to ensure that the model performs well on unseen data and is not overfitting or underfitting

How Cross-Validation Works

Split the Data

- The dataset is divided into k equal (or nearly equal) parts or "folds"

Training and Testing

- In each iteration, $k-1$ folds are used for training, and the remaining fold is used for testing (validation)
- This process is repeated k times, with each fold used exactly once as the testing set

Aggregate Results

- The performance metric (e.g., accuracy, precision, recall) is averaged across all k iterations to get a more reliable estimate of the model's performance

Why Cross-Validation is Used in Machine Learning

- Improves Model Generalization
- Provides a Robust Performance Estimate
- Efficient Use of Data

Types of Cross-Validation

- k-Fold Cross-Validation
 - The most common type
- Stratified k-Fold Cross-Validation
 - Ensures each fold has a proportional representation of each class,
 - useful for imbalanced datasets
- Leave-One-Out Cross-Validation (LOOCV)
 - Each data point is used once as the validation set
 - the remaining data points are used for training
 - Very computationally expensive but provides an unbiased estimate

Evaluation Metrics

How to evaluate a prediction

- $h(x) = y \text{ hat}$
- Y is the actual value
- Error if both are different
- To measure error
 - Absolute error
 - on a single training example $|h(x)-y|$
 - On multiple training example $1/n \sum |h(x)-y|$
 - Sum of squares error
 - $1/n \sum (h(x)-y)^2$
- These absolute and sum of squares error are specially useful in case of regression
- For classification we look at the number of misclassifications
 - $1/n \sum \delta(h(x), y)$ where $\delta(h(x), y)$ is 0 if $h(x)$ and y are same and 1 if they are same

Confusion matrix

- In case of classification problem it is helpful
- For three class classification
 - 3x3 matrix
 - Diagonal entries are where the learning algorithm is giving the correct results
 - Non-diagonal entries are where the learning algorithm is giving correct results

		true class	
		Actually Positive (1)	Actually Negative (0)
hypothesised class	Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
	Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Other evaluation metrics

- Accuracy
 - Correct examples out of all examples
 - $(TP + TN) / (TP + TN + FP + FN)$
- Precision
 - Accuracy of positive predictions
 - $TP / (TP + FP)$
 - A trivial way to have perfect precision is
 - To make one single positive prediction and ensure it is correct precision = $(1/1 = 100\%)$
- Recall/ Sensitivity/ true positive rate (TPR)
 - Ratio/proportion of positive instances that are correctly predicted by the classifier.
 - $TP / (TP + FN) = (TP/P)$
- Specificity/ True negative rate (TNR)
 - Ratio/proportion of true negatives that are correctly predicted by the model
 - $TN / (TN + FP) = (TN/N)$
- False Positive rate (FPR)
 - Ratio/proportion of actual negative cases that are incorrectly predicted as positive by the classifier
 - $FP / (FP + TN) = (FP/N)$

		true class	
		Actually Positive (1)	Actually Negative (0)
hypothesised class	Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
	Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

P

N

Other evaluation metrics

- F1 score
 - Combines precision and recall to a simple way to compare two classifiers.
 - Harmonic mean of precision and recall.
 - favours classifiers that have similar precision and recall.
- Whereas the regular mean treats all values equally, the harmonic mean gives much more weight to low values.
- As a result, the classifier will only get high F1 score if both precision and recall are high.
- $F1 = \frac{2}{(1/\text{Precision}) + (1/\text{Recall})} = \frac{2 \cdot (\text{Precision} \cdot \text{Recall})}{\text{Precision} + \text{Recall}}$

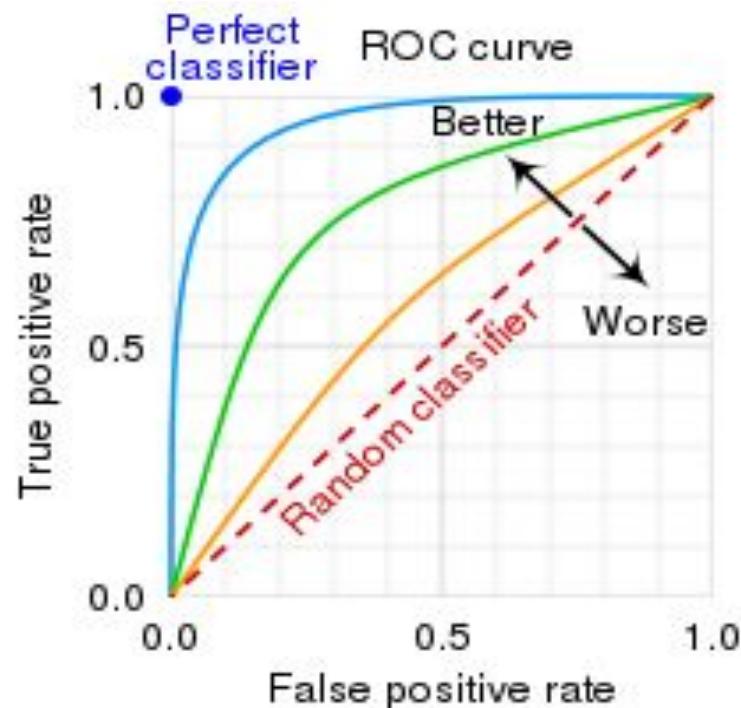
Precision-Recall tradeoff

- F1 favours classifiers that have similar precision and recall.
- Many times we don't want both Precision and Recall to be high
- Sometimes we mostly care about precision, and in other contexts we really care about recall.
- Example:
 - classifier to detect videos that are safe for kids
 - We want a classifier with High precision and don't want a classifier with high recall (we want 0 False positives) (in this case positive is –safe video)
 - Classifier to detect shoplifters on surveillance images
 - Classifier with low precision (the security guard will get a few false alerts) but high recall (almost all shoplifters will get caught) would be fine (We want 0 False negative)
- Unfortunately, you can't have it both ways: increasing one will reduce the other.

Receiver Operating Characteristic (ROC) curve

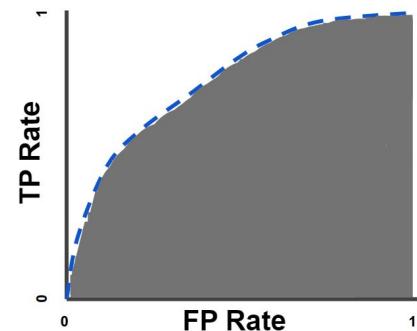
- The ROC curve is generated by plotting the TPR (sensitivity or recall) against the FPR at various classification threshold settings
- Classification threshold
 - the model usually assigns a probability score or a continuous value to each instance, indicating the likelihood of it belonging to one of the classes.
 - To make a binary decision, this continuous value needs to be converted into a discrete decision, typically by choosing a threshold.
 - For example, if the threshold is set at 0.5,
 - any instance with a predicted probability above 0.5 would be classified as positive, and
 - any instance with a predicted probability below 0.5 would be classified as negative.
 - Adjusting this threshold affects the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) and consequently affects the shape of the ROC curve

Receiver Operating Characteristic (ROC) curve



AUC (Area under the ROC Curve)

- A perfect classifier would have an ROC curve that passes through the top-left corner of the plot ($TPR = 1$, $FPR = 0$), indicating high sensitivity and specificity.
- The area under the ROC curve (AUC) is a commonly used metric to quantify the overall performance of the classifier.
- AUC provides an aggregate measure of performance across all possible classification thresholds.
- AUC ranges in value from 0 to 1.
- A perfect classifier would have an AUC of 1
- A model whose predictions are 100% wrong has an AUC of 0.0
- A random classifier would have an AUC of 0.5.
- A higher AUC indicates better overall performance of the classifier across all possible thresholds.



K-Means Clustering Algorithm

K-means Algorithm

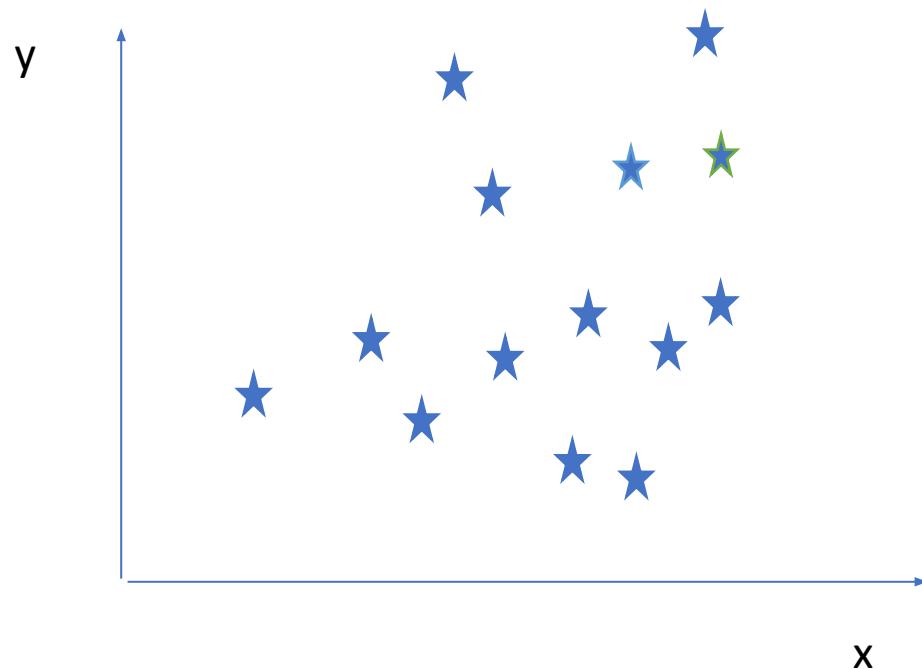
- Discovers categories/groups in the dataset

K-means Algorithm

- Step1: Choose the number K of clusters
- Step2: Select at random k points, the centroids (not necessarily from the dataset)
- Step3: Assign each data point to the closest centroid? that forms k clusters
- Step4: Compute and place the new centroid of each cluster
- Step5: Reassign each data point to the new closest centroid. If any reassignment took place, go to step4, otherwise Stop (the model is ready)

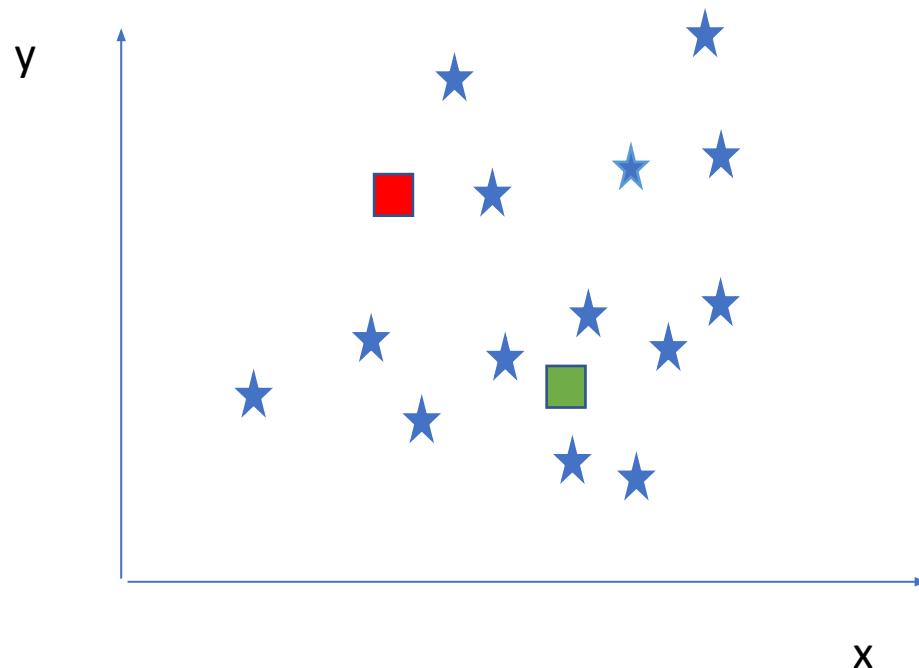
Dataset

Step1: Choose the number K of clusters K=2



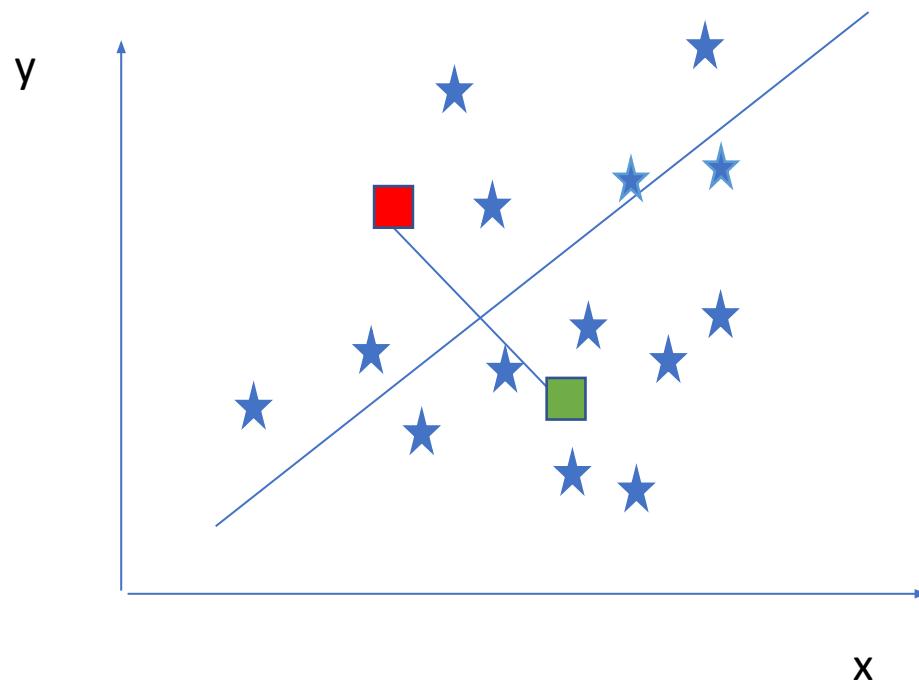
Dataset

Step2: Select at random k points, the centroids (not necessarily from the dataset)



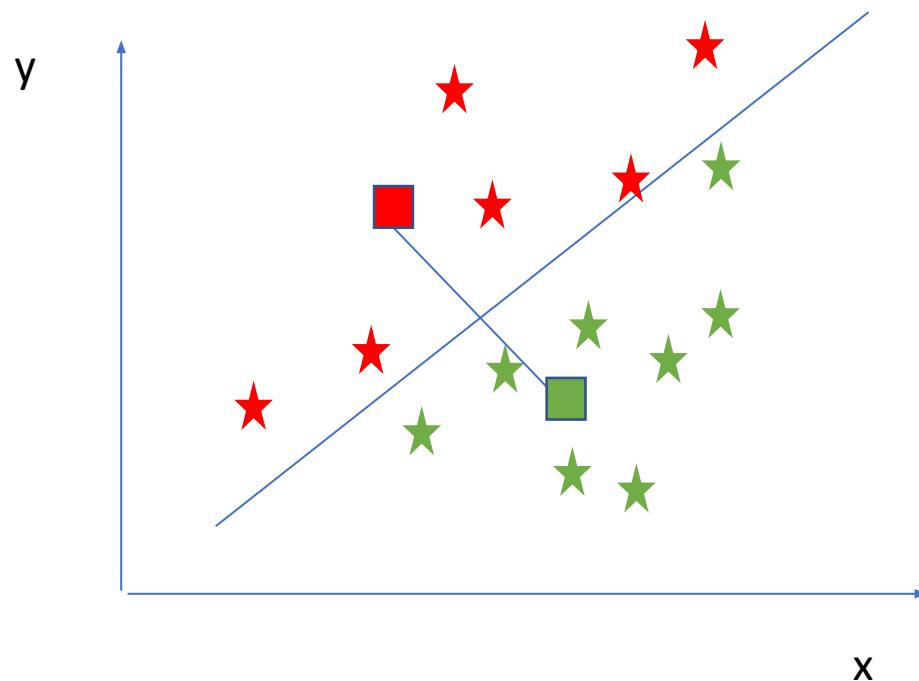
Dataset

Step3: Assign each data point to the closest centroid[¶] that forms k clusters



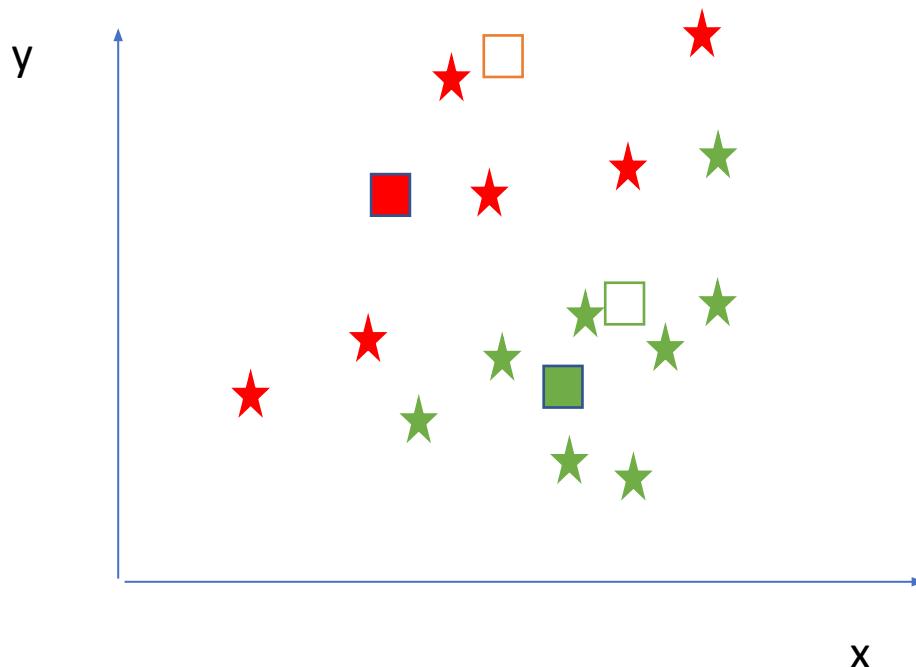
Dataset

Step3: Assign each data point to the closest centroid[¶] that forms k clusters



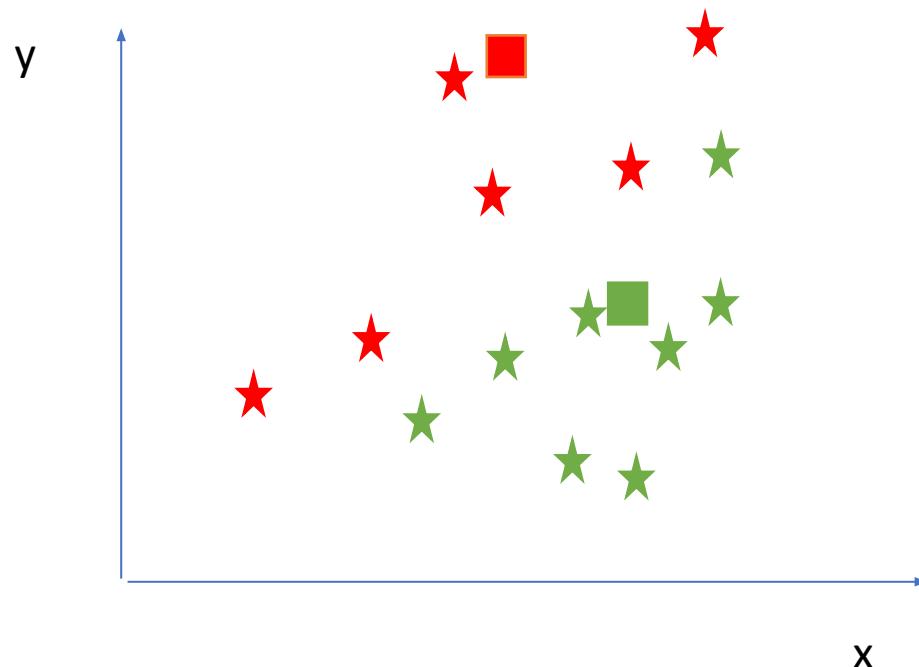
Dataset

Step5: Reassign each data point to the new closest centroid. If any reassignment took place, go to step4, otherwise Stop (the model is ready)



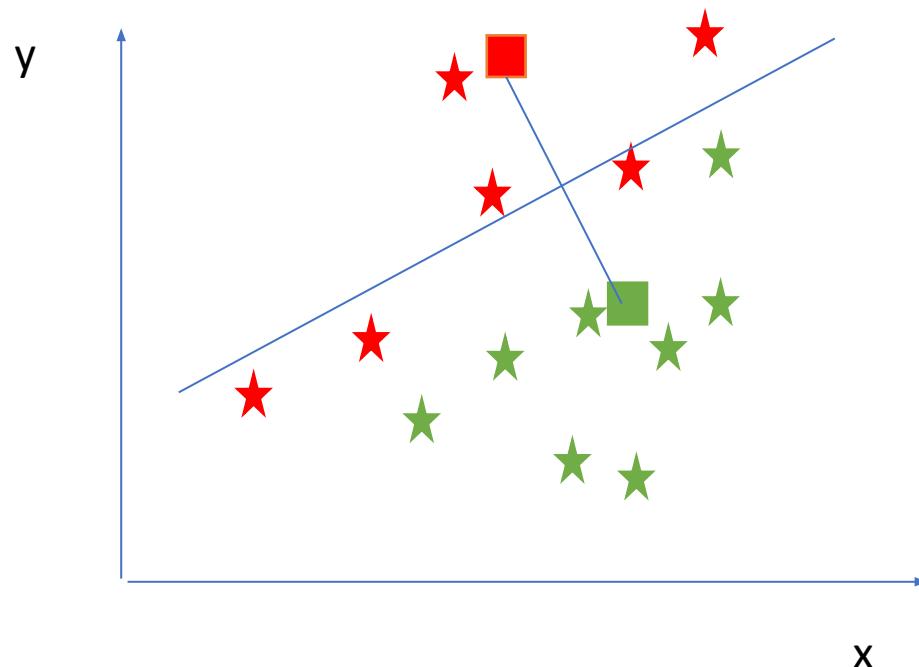
Dataset

Step5: Reassign each data point to the new closest centroid. If any reassignment took place, go to step4, otherwise Stop (the model is ready)



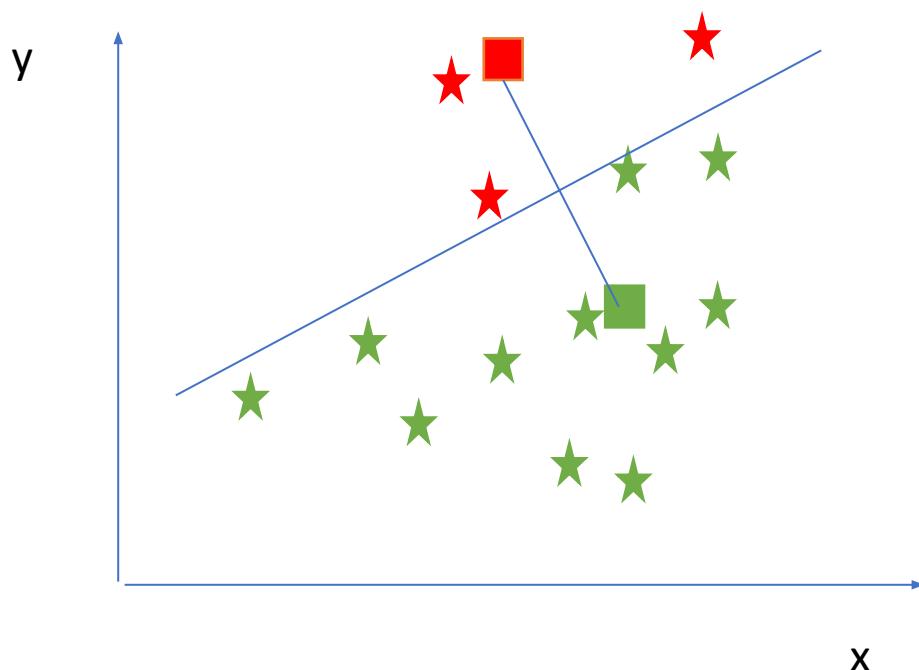
Dataset

Step5: Reassign each data point to the new closest centroid. If any reassignment took place, go to step4, otherwise Stop (the model is ready)



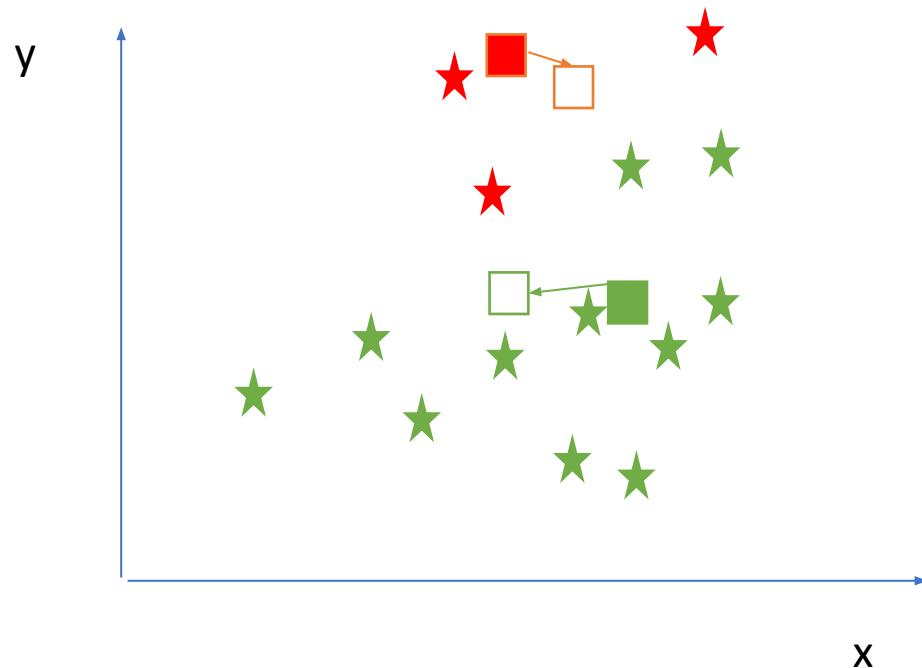
Dataset

Step5: Reassign each data point to the new closest centroid. If any reassignment took place, go to step4, otherwise Stop (the model is ready)



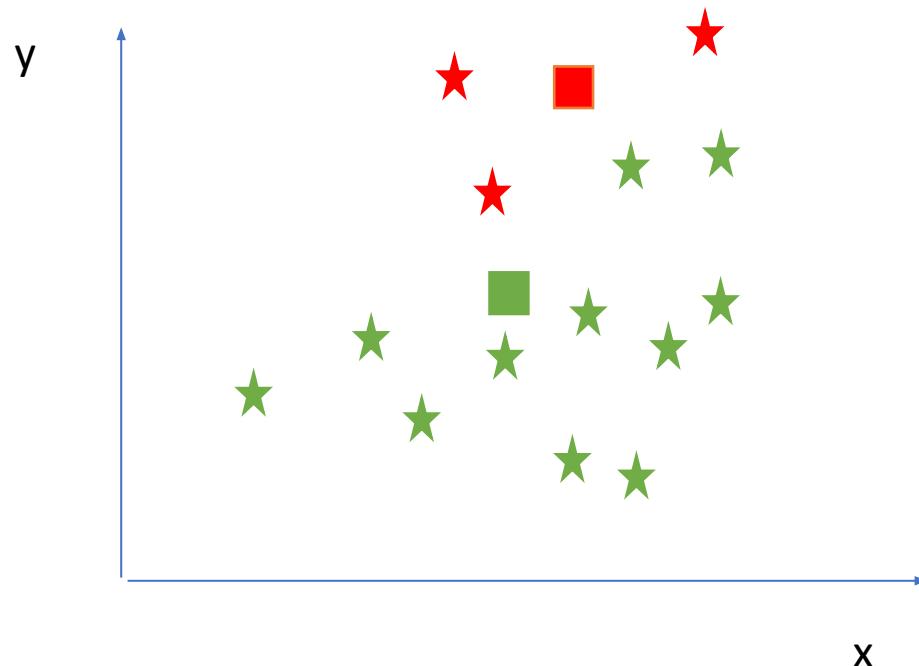
Dataset

Step4: Compute and place the new centroid of each cluster



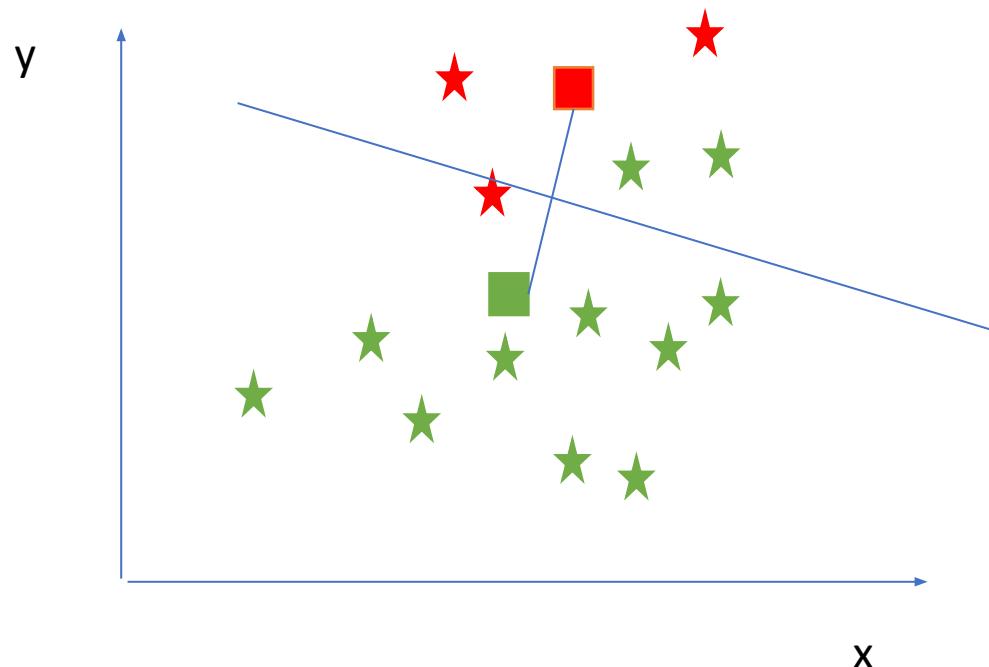
Dataset

Step4: Compute and place the new centroid of each cluster



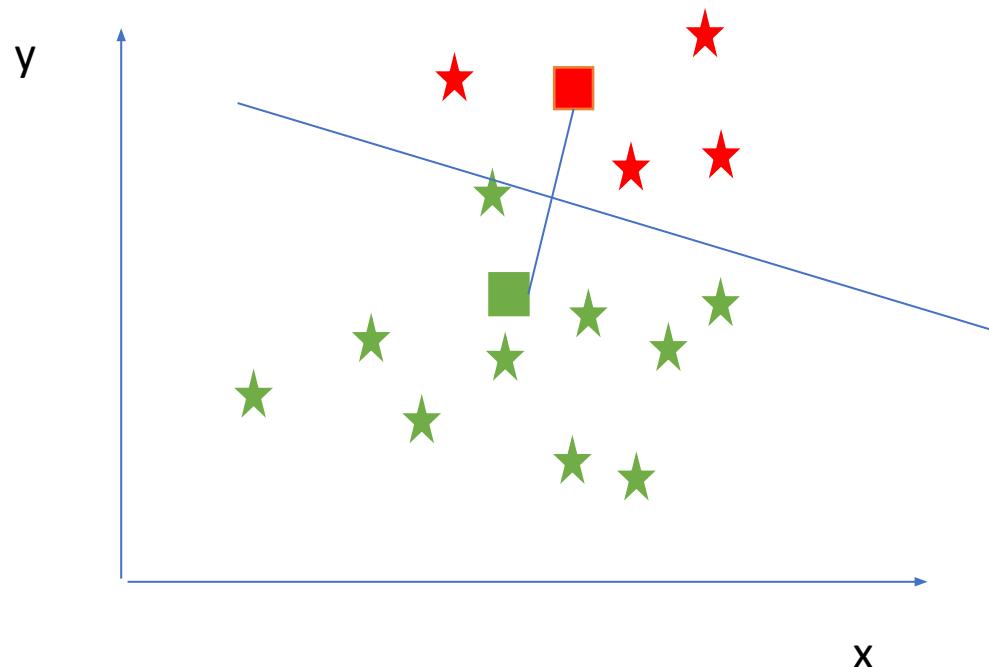
Dataset

Step5: Reassign each data point to the new closest centroid. If any reassignment took place, go to step4, otherwise Stop (the model is ready)



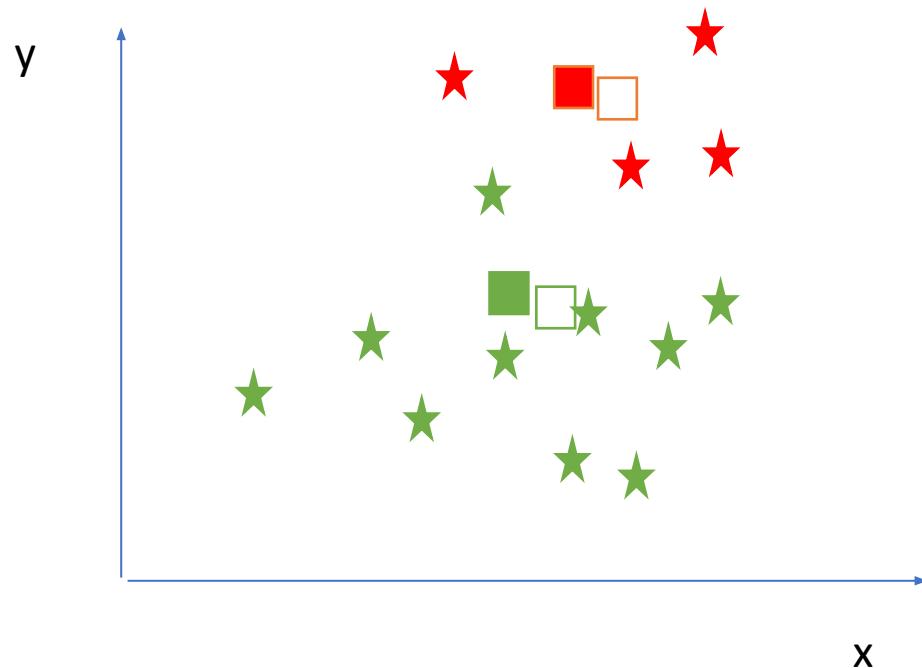
Dataset

Step5: Reassign each data point to the new closest centroid. If any reassignment took place, go to step4, otherwise Stop (the model is ready)



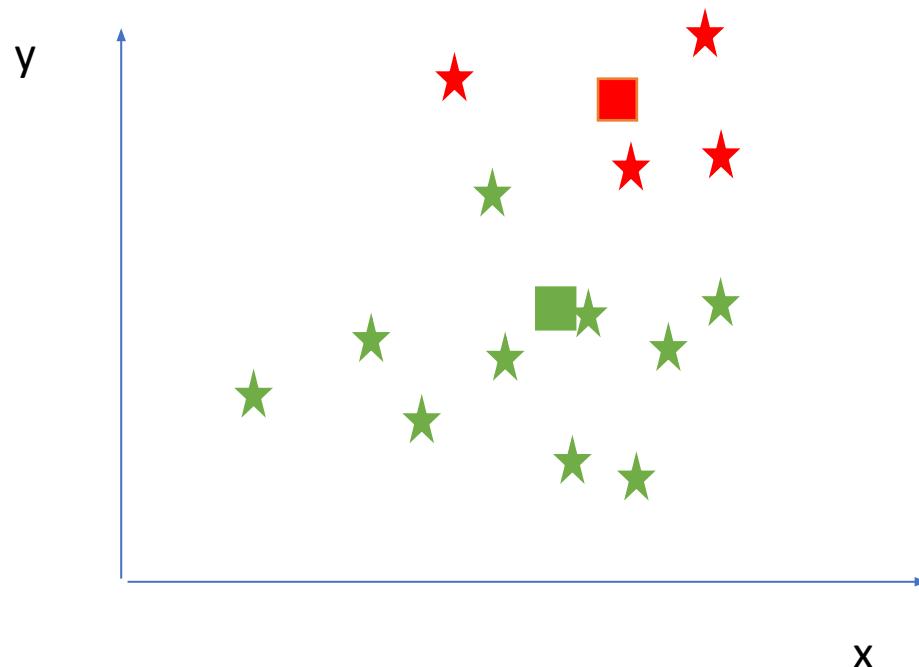
Dataset

Step4: Compute and place the new centroid of each cluster



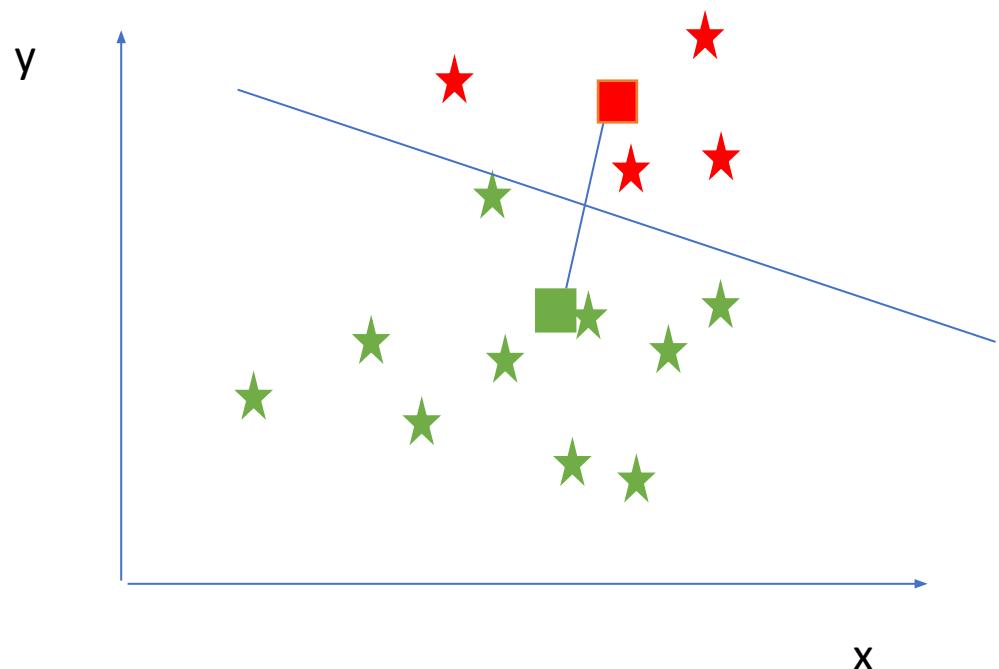
Dataset

Step4: Compute and place the new centroid of each cluster



Dataset

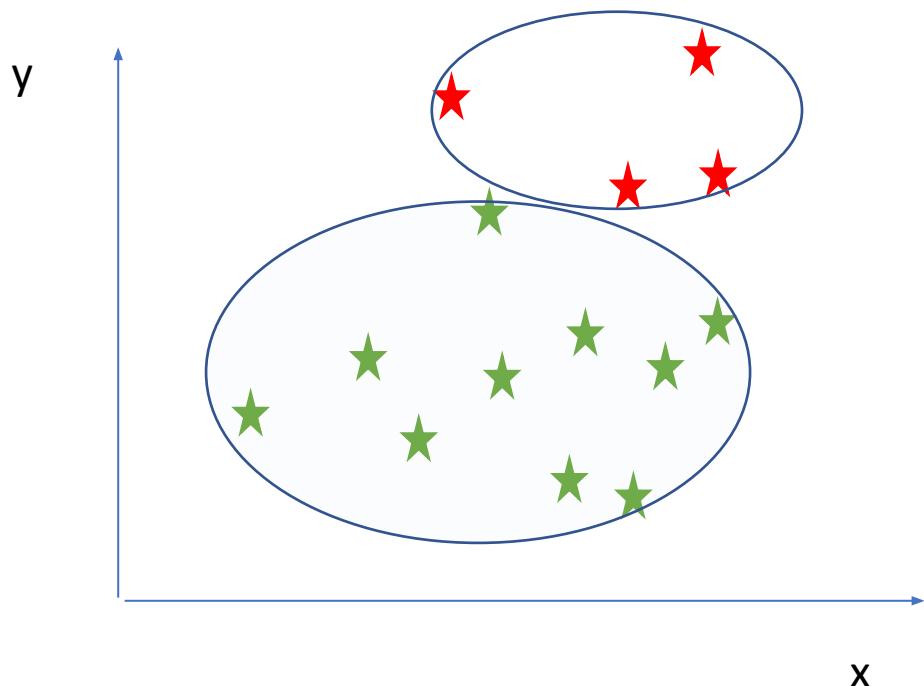
Step5: Reassign each data point to the new closest centroid. If any reassignment took place, go to step4, otherwise Stop (the model is ready)



Since no reassignment took place, Stop (the model is ready)

Dataset

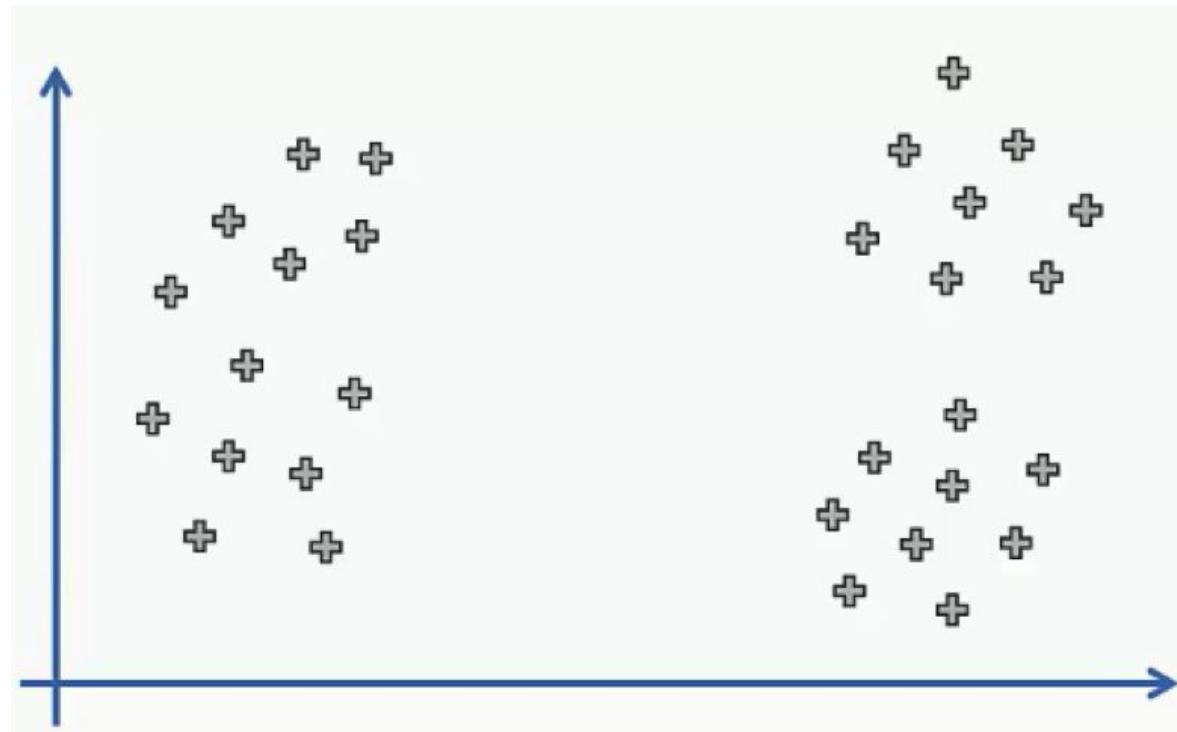
Step5: Reassign each data point to the new closest centroid. If any reassignment took place, go to step4, otherwise Stop (the model is ready)



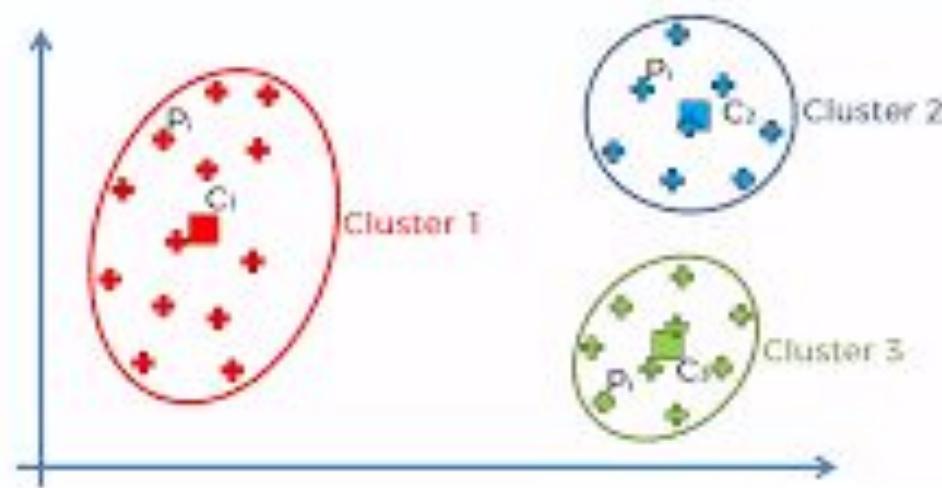
Since no reassignment took place, Stop (the model is ready)

- relatively efficient with time complexity $O(nkt)$ where-
- n = number of instances
- k = number of clusters
- t = number of iterations

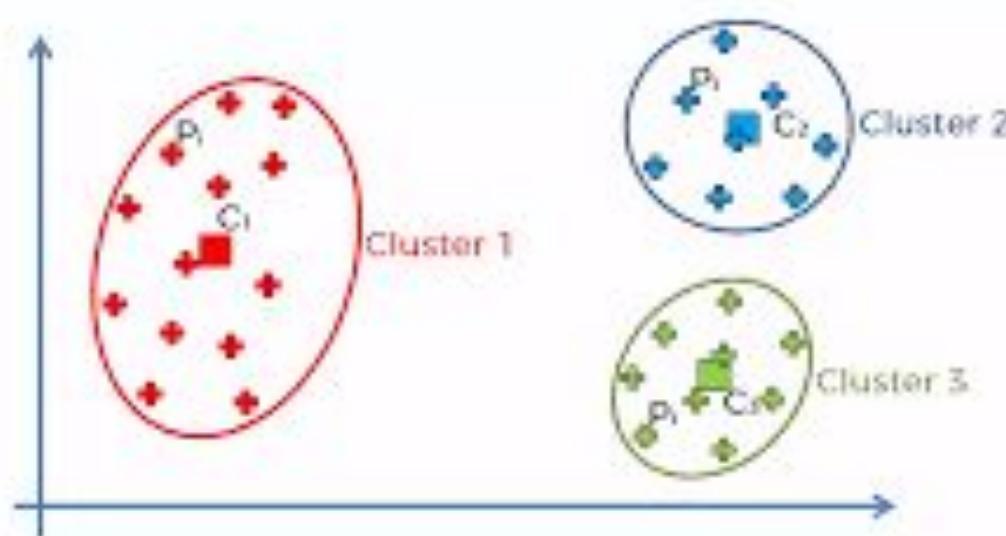
Choosing the correct number of clusters



Choosing the correct number of clusters



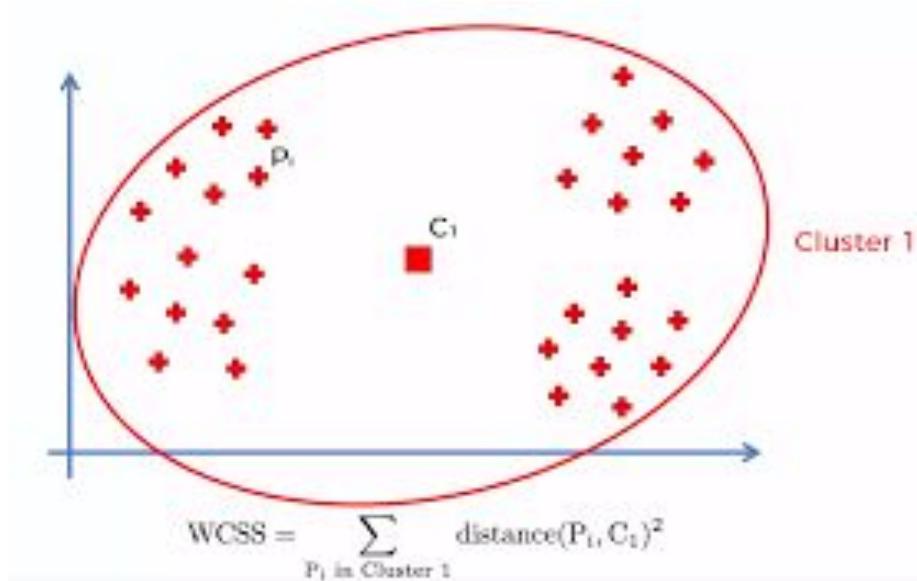
Choosing the correct number of clusters



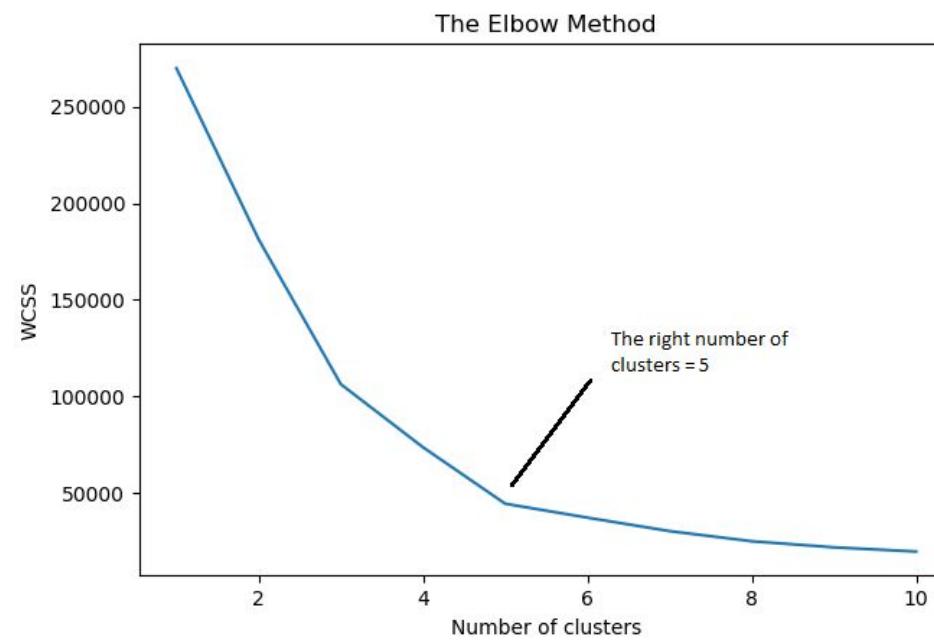
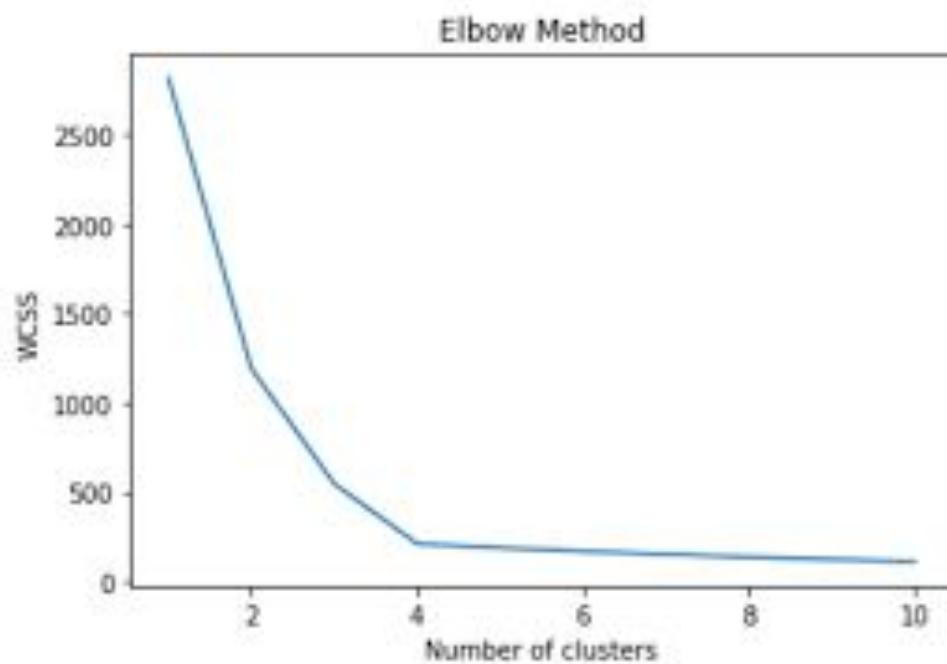
$$WCSS = \sum_{P_i \text{ in Cluster 1}} \text{distance}(P_i, C_1)^2 + \sum_{P_i \text{ in Cluster 2}} \text{distance}(P_i, C_2)^2 + \sum_{P_i \text{ in Cluster 3}} \text{distance}(P_i, C_3)^2$$

With in clusters sum
of squares

Choosing the correct number of clusters



Choosing the correct number of clusters



Distance Metrics in Machine Learning

To deal with continuous or numerical variables

1. Euclidean Distance
2. Manhattan Distance
3. Minkowski Distance

To deal with categorical variables

4. Hamming Distance

Formula for Euclidean Distance

$$d = ((p_1 - q_1)^2 + (p_2 - q_2)^2)^{1/2}$$

$$D_e = \left(\sum_{i=1}^n (p_i - q_i)^2 \right)^{1/2}$$

Where,

- n = number of dimensions
- pi, qi = data points

Manhattan Distance

- is the sum of absolute differences between points across all the dimensions

$$d = |p_1 - q_1| + |p_2 - q_2|$$

$$D_m = \sum_{i=1}^n |p_i - q_i|$$

Where,

- n = number of dimensions
- pi, qi = data points

Minkowski Distance

- Minkowski Distance is the generalized form of Euclidean and Manhattan Distance.

$$D = \left(\sum_{i=1}^n |p_i - q_i|^p \right)^{1/p}$$

Hamming Distance

- measures the similarity between two strings of the same length
- is the number of positions at which the corresponding characters are different.

Example

Consider the below data set which has the values of the data points

Documents (Data Points)	W1 (x-axis)	W2 (y-axis)
D1	2	0
D2	1	3
D3	3	5
D4	2	2
D5	4	6

Apply K-means algorithm to find two clusters.

Iteration 1

- Step 1: k=2
- Step 2: randomly choose two initial points as the centroids- D2 and D4 are the centroids.
- Step 3: We need to calculate the distance between the initial centroid points with other data points.

Distance between D1 and D2	Distance between D1 and D4
$\begin{aligned} & \sqrt{(2-1)^2 + (0-3)^2} \\ &= \sqrt{(1)^2 + (3)^2} \\ &= \sqrt{1+9} \\ &= \sqrt{10} = 3.17 \end{aligned}$	$\begin{aligned} & \sqrt{(2-2)^2 + (0-2)^2} \\ &= \sqrt{(0)^2 + (-2)^2} \\ &= \sqrt{0+4} \\ &= \sqrt{4} = 2 \end{aligned}$

Step 3

Documents (Data Points)	Distance between D2 and other data points	Distance between D4 and other data points
D1	3.17	2.0
D3	2.83	3.17
D5	4.25	4.48

- **Step 3:** Next, we need to group the data points which are closer to centroids.
- **Cluster 1: (D1, D4) Cluster 2: (D2, D3, D5)**
- **Step 4:** Now, we calculate the mean values of the clusters created and the new centroid values will these mean values and centroid is moved along the graph.

Clusters	Mean value of data points along x -axis	Mean value of data points along y -axis
D1, D4	2.0	1.0
D2, D3, D5	2.67	4.67

- new centroid for cluster 1 is (2.0, 1.0) and for cluster 2 is (2.67, 4.67)

Iteration 2

- **Step 3:** Again the values of Euclidean distance is calculated from the new centroids.

Documents (Data Points)	Distance between centroid of cluster 1 and data points	Distance between centroid of cluster 2 and data points
D1	1.0	4.72
D2	2.24	2.37
D3	4.13	0.47
D4	1	2.76
D5	5.39	1.89

- Cluster 1: (D1, D2, D4) Cluster 2: (D3, D5)

- **Step 4:** Calculate the mean values of new clustered group

Clusters	Mean value of data points along x -axis	Distance between D4 and other data points
D1, D2, D4	1.67	1.67
D3, D5	3.5	5.5

- new centroid value as following:
- **cluster 1 (D1, D2, D4) - (1.67, 1.67) and cluster 2 (D3, D5) - (3.5, 5.5)**
- Iteration 3...

Importing the libraries

- `import numpy as np`
- `import matplotlib.pyplot as plt`
- `import pandas as pd`

Importing the dataset

- dataset = pd.read_csv('Mall_Customers.csv')
- X = dataset.iloc[:, [3, 4]].values

Using the elbow method to find the optimal number of clusters

```
• from sklearn.cluster import KMeans  
• wcss = []  
• for i in range(1, 11):  
•     kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)  
•     kmeans.fit(X)  
•     wcss.append(kmeans.inertia_)  
• plt.plot(range(1, 11), wcss)  
• plt.title('The Elbow Method')  
• plt.xlabel('Number of clusters')  
• plt.ylabel('WCSS')  
• plt.show()
```

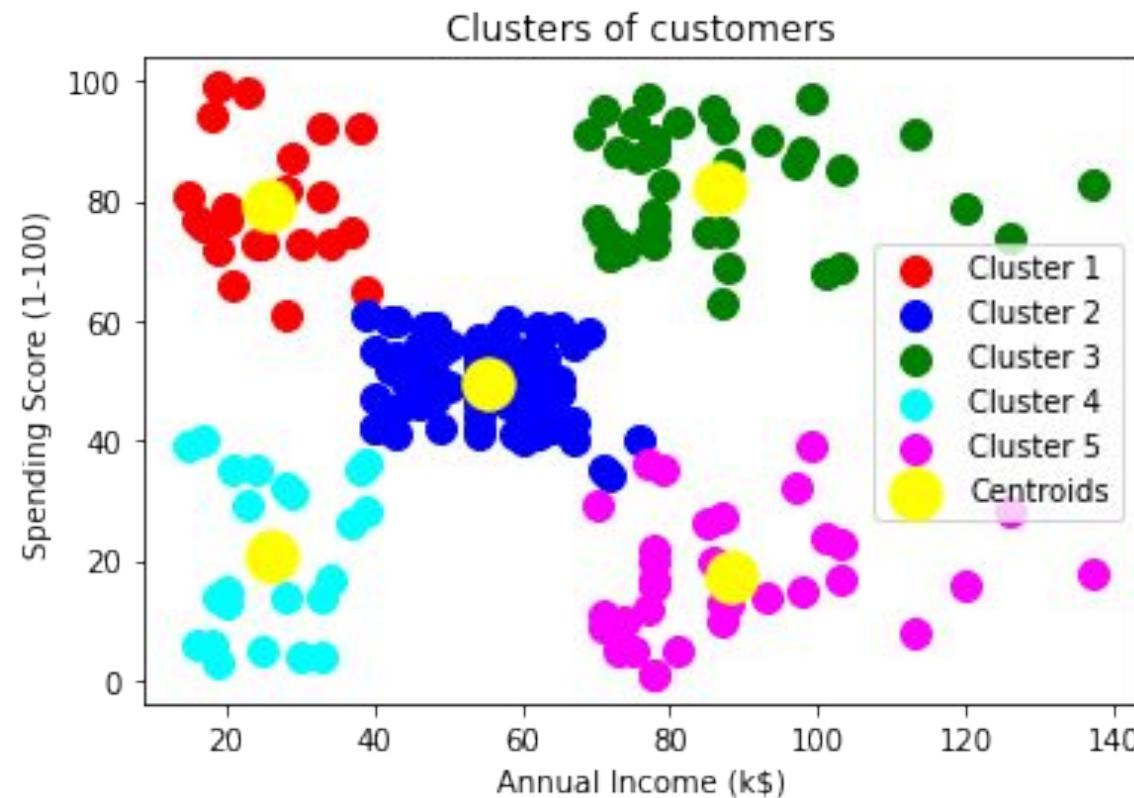
Training the K-Means model on the dataset

- `kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)`
- `y_kmeans = kmeans.fit_predict(X)`

Visualising the clusters

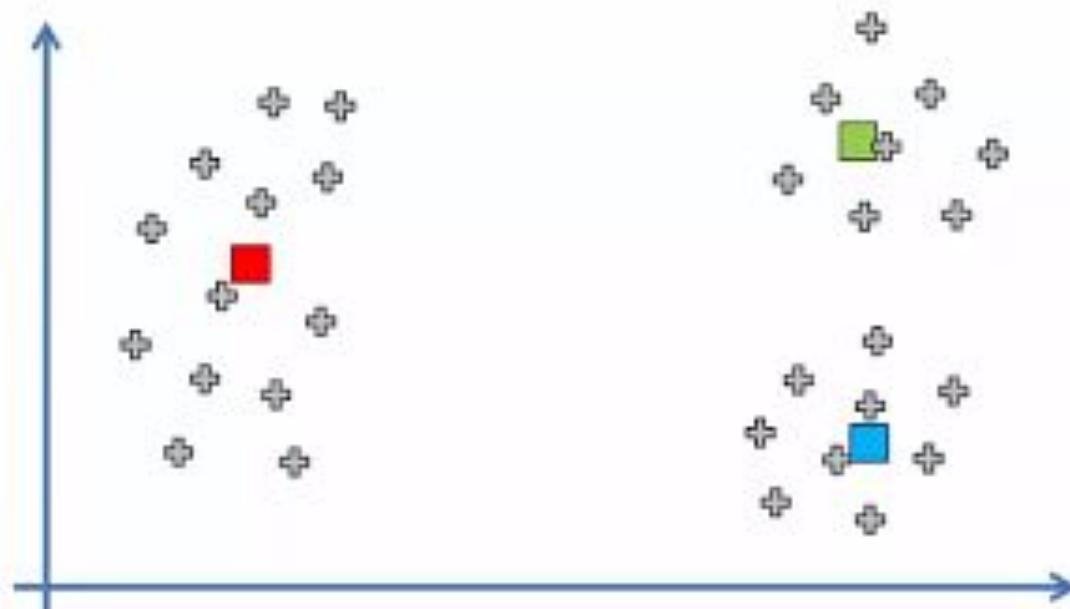
- `plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')`
- `plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')`
- `plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')`
- `plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')`
- `plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')`
- `plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids')`
- `plt.title('Clusters of customers')`
- `plt.xlabel('Annual Income (k$)')`
- `plt.ylabel('Spending Score (1-100)')`
- `plt.legend()`
- `plt.show()`

Visualising the clusters

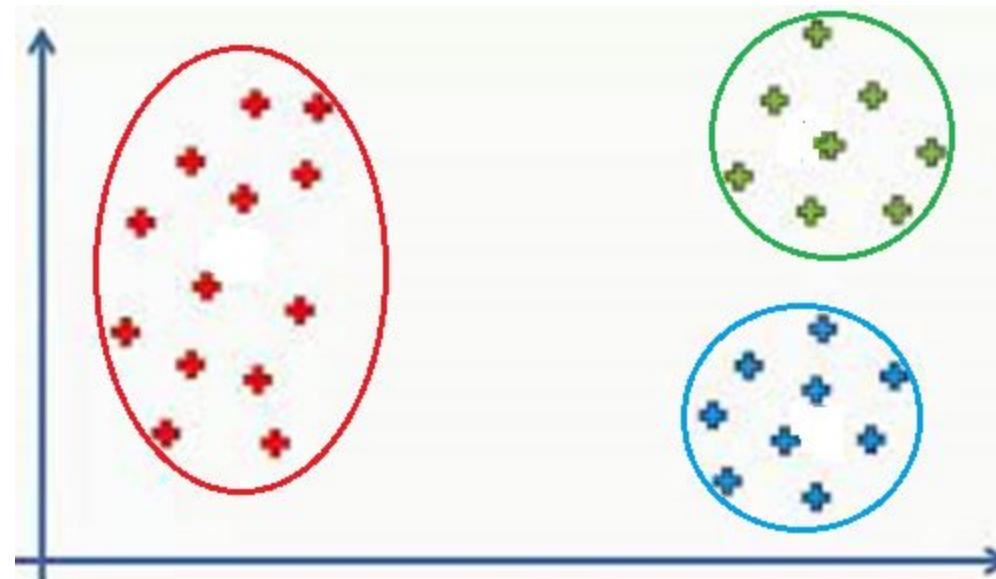


K-means++

Random Initialization Trap



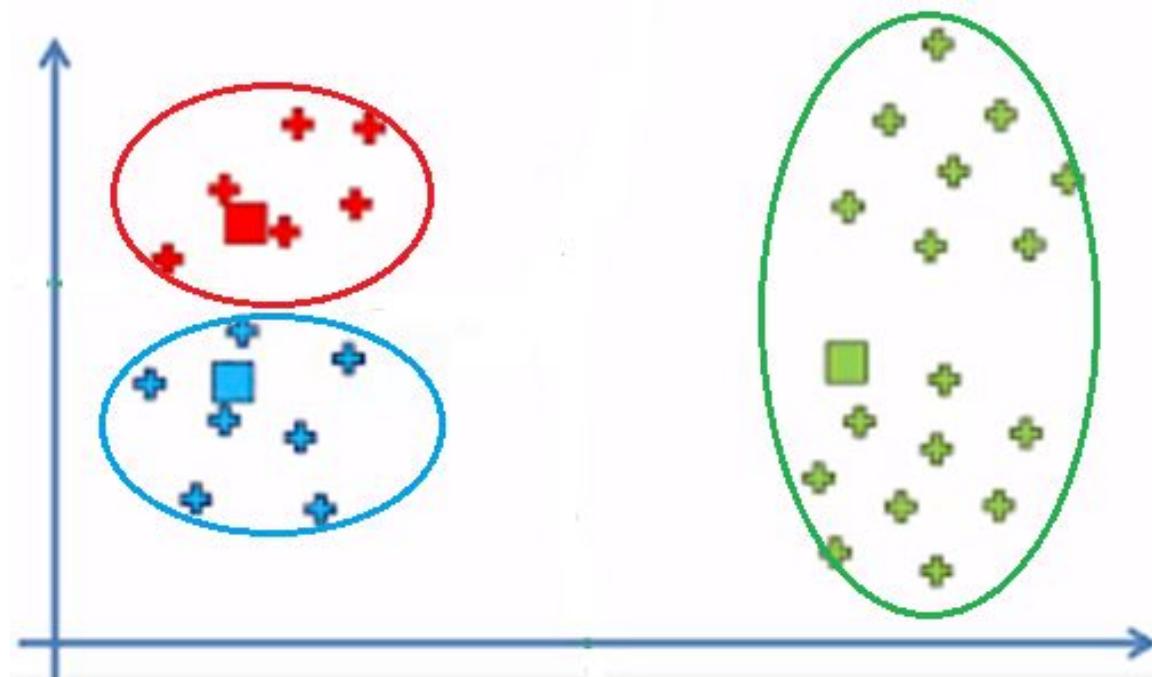
Clusters Found



Another random Initialization



Different Clusters found



Solution

- K-means++
- The K-means++ algorithm is an extension of the popular k-means algorithm, which randomly selects initial cluster centroids.
- K-means++, on the other hand, uses a more sophisticated approach to select the initial centroids.
- Intuition-
 - By selecting the initial centroids in a smart way, we can improve the overall performance of the algorithm.
 - By choosing the initial centroids that are far apart from each other, we increase the chances of capturing the underlying structure of the data and producing better clusters.

The steps involved in the K-means++

1. Initialize the first centroid randomly from the dataset.
2. For each data point, compute the minimum squared distance (i.e., the distance squared between a data point and the nearest centroid) and store it in a list.
3. Randomly select the next centroid from the dataset using a probability distribution that is proportional to the minimum squared distance of each data point from the nearest centroid.
 - This means that the more distant a point is from the nearest centroid, the more likely it is to be selected as the next centroid.
4. Repeat steps 2 and 3 until all K centroids have been selected.
5. Run the standard K-means algorithm with the K initial centroids obtained from the above steps.

After finding probability of each points how to calculated cumulative distribution function

1. Sort the probabilities of each data point in ascending order.
2. Normalize the sorted probabilities so that they add up to 1. This can be done by dividing each probability by the sum of all probabilities.
3. Calculate the cumulative sum of the normalized probabilities. This can be done by adding up the normalized probabilities from the first data point to the last data point.
4. The resulting cumulative sum represents the CDF of the probability distribution.

How to select a random point based on the probability distribution calculated

- To select a random point based on the probability distribution calculated in K-means++, we can use a weighted sampling approach.
- The steps to do so are:
 1. Calculate the cumulative distribution function (CDF) of the probability distribution obtained in K-means++. This can be done by summing up the probabilities from the first point to the last point.
 2. Generate a random number between 0 and 1.
 3. Use the CDF to find the data point whose cumulative probability is greater than or equal to the random number generated in step 2. This can be done using binary search or linear search algorithms.
 4. Select the data point found in step 3 as the next centroid.

Example

- Suppose we have a dataset of 10 two-dimensional points:
 - $[(1, 1), (1, 2), (2, 1), (2, 2), (3, 3), (3, 4), (4, 3), (4, 4), (5, 5), (5, 6)]$
- We want to cluster these points into two clusters using K-means++.
- The steps of K-means++ are as follows:
 1. Select the first centroid at random from the dataset. Let's say we select the point $(2, 2)$ as the first centroid.
 2. Calculate the squared distance between each point and the first centroid.
 3. Assign each point a probability proportional to its squared distance from the first centroid.
 4. Normalize the probabilities so that they add up to 1.
 - Squared distances:
 $[2, 1, 1, 2, 13, 18, 13, 18, 32, 37]$
 - Probabilities:
 $[0.0154, 0.0077, 0.0077, 0.0154, 0.2462, 0.3385, 0.2462, 0.3385, 0.5846, 0.6769]$

- 3. Select the second centroid by randomly choosing a point from the dataset, with the probability of each point being proportional to its squared distance from the first centroid.
- Suppose we generate a random number between 0 and 1 and it turns out to be 0.7.
- We can use the CDF to find the data point whose cumulative probability is greater than or equal to 0.7:
 - CDF:
 - [0.0154, 0.0231, 0.0308, 0.0462, 0.2923, 0.6308, 0.877, 1.0, 1.0, 1.0]
 - Select data point at index 7: (4, 4)

4. Perform K-means clustering with the two centroids $(2, 2)$ and $(4, 4)$ to assign each point to its nearest centroid, and update the centroids based on the mean of the points assigned to each cluster.
5. Repeat steps 2-4 to select additional centroids, until the desired number of clusters is reached.
 - The final result after two iterations of K-means would be:
 - Cluster 1: $[(1, 1), (1, 2), (2, 1), (2, 2), (3, 3), (4, 3)]$
 - Cluster 2: $[(3, 4), (4, 4), (5, 5), (5, 6)]$

K-means++

- It iteratively assigns each data point to its nearest centroid and updates the centroid location based on the new cluster assignments.
- The k-means++ algorithm has been shown to perform better than the k-means algorithm on several datasets, especially when the number of clusters is not known a priori.

Hierarchical Clustering

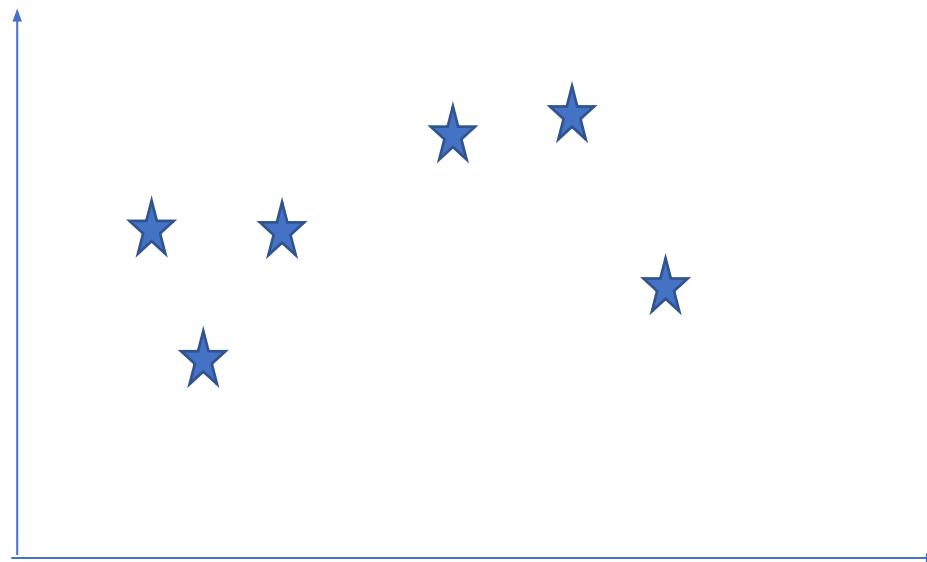
Hierarchical Clustering (HC)

- Results of HC are sometimes very similar to K-means algorithm
- Types of HC
 - Agglomerative
 - divisive

Agglomerative HC

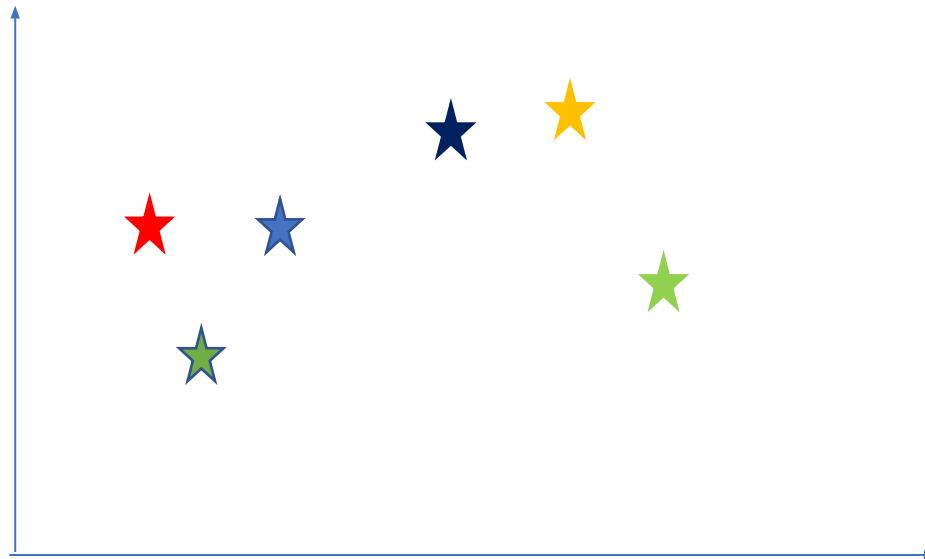
- Step 1: Make each data point a single-point cluster \square that forms N clusters
- Step 2: Take the two closest data points and make them one cluster \square that forms N-1 clusters
- Step 3: Take the two closest clusters and make them one cluster \square that forms N-2 clusters
- Step 4: Repeat Step 3 until there is only one cluster

Example (Agglomerative HC)



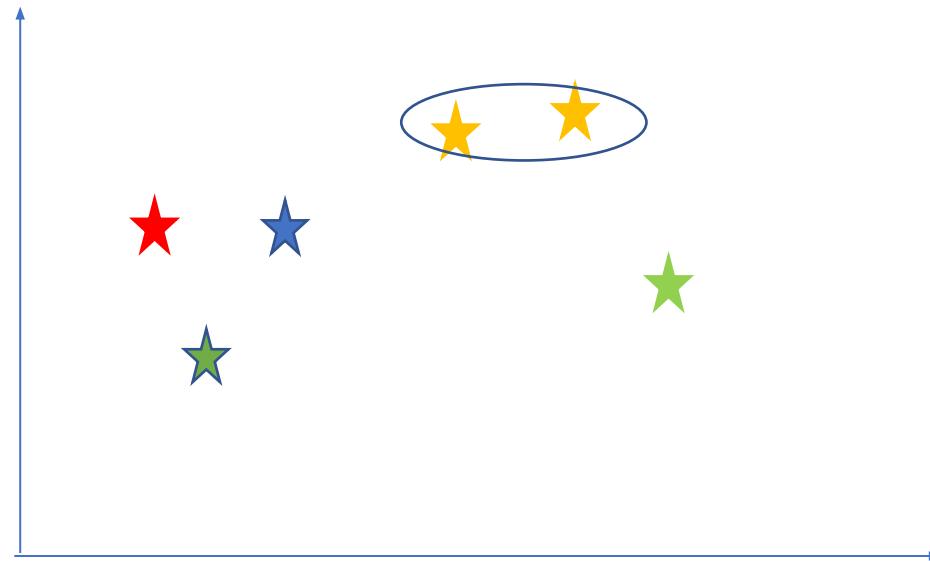
Example (Agglomerative HC)

Step1: Make each data point a single-point cluster that forms 6 clusters



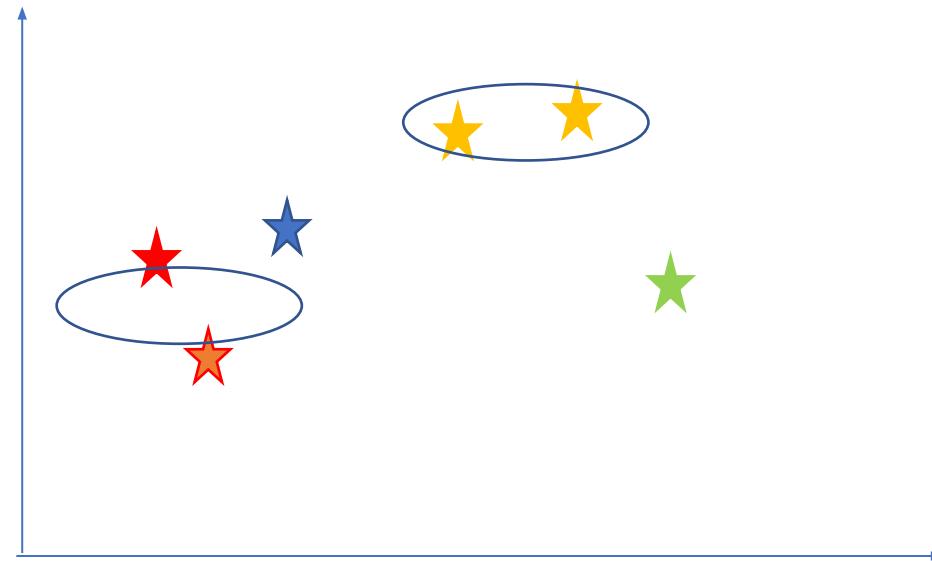
Example (Agglomerative HC)

Step 2: Take the two closest data points and make them one cluster  that forms 5 (N-1) clusters



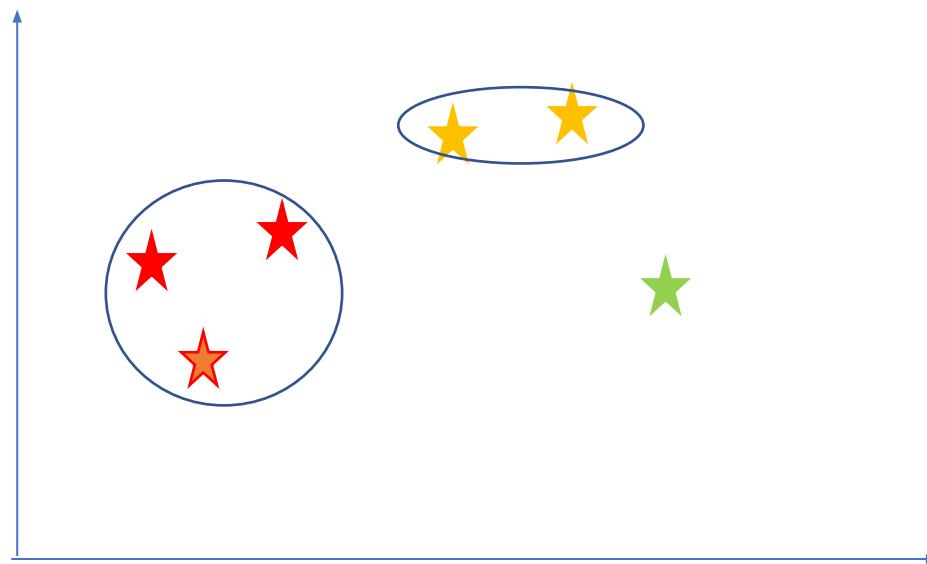
Example (Agglomerative HC)

Step 3: Take the two closest clusters and make them one cluster  that forms 4 clusters



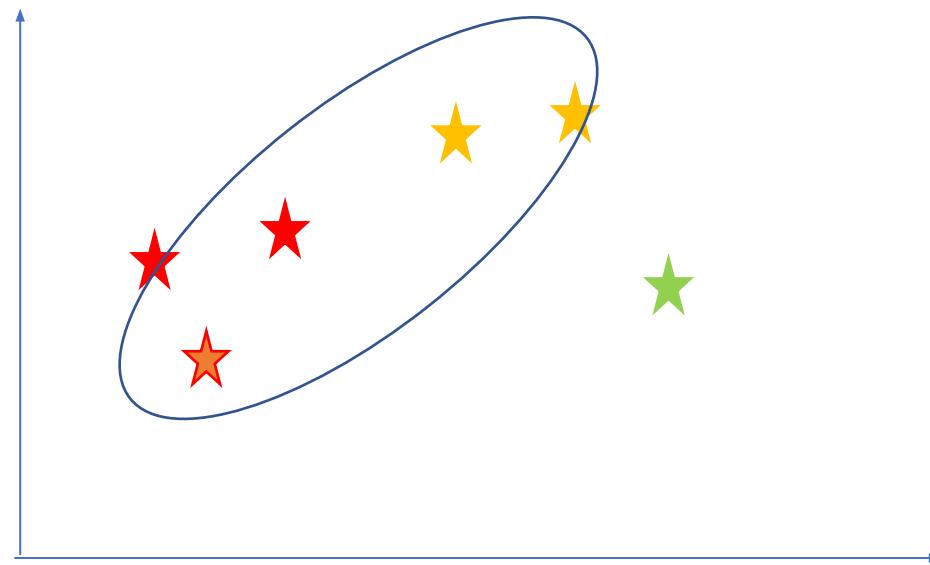
Example (Agglomerative HC)

Step 4: Repeat Step 3 until there is only one cluster



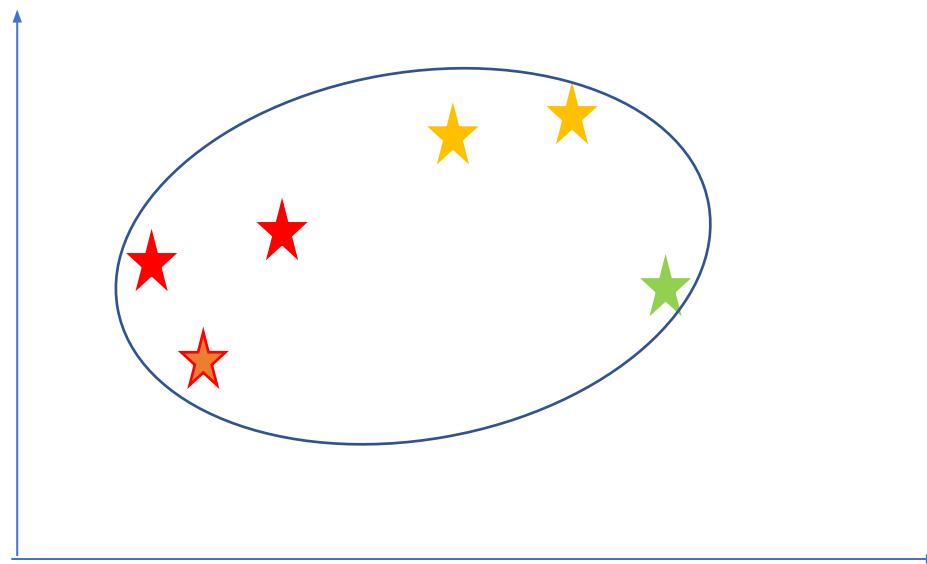
Example (Agglomerative HC)

Step 4: Repeat Step 3 until there is only one cluster



Example (Agglomerative HC)

Step 4: Repeat Step 3 until there is only one cluster



Distance/Some examples for agglomerative clustering

- Distance between two data points
 - Euclidean distance or any other distance
- Distance between two clusters
 - *single-linkage*
 - complete linkage
 - average linkage
 - *centroid linkage*
 - Ward's method

Single-linkage distance

- Distance between two closest point
- $\text{Dis}(C1, C2) = \text{Min dist}(P_i, P_j)$ such that $P_i \in C1 \text{ & } P_j \in C2$
- To calculate the single linkage distance between two clusters, we first need to calculate the distance between all pairs of points in the two clusters.
- Then we select the minimum distance between any two points in the two clusters as the single linkage distance between the clusters.
- Single linkage tends to produce elongated, chain-like clusters, and can be sensitive to noise and outliers in the data.

Complete linkage distance

- Distance between two furthest point
- $\text{Dis}(C1, C2) = \text{Min dist}(P_i, P_j)$ such that $P_i \in C1$ & $P_j \in C2$
- To calculate the complete linkage distance between two clusters, we first need to calculate the distance between all pairs of points in the two clusters.
- Then we select the maximum distance between any two points in the two clusters as the complete linkage distance between the clusters.
- Complete linkage tends to produce compact, spherical clusters, and is less sensitive to noise and outliers in the data compared to single linkage. However, it can be sensitive to the presence of elongated clusters in the data.

Average linkage (Group Average)

- The average linkage method calculates the distance between two clusters as the average distance between all pairs of points, one from each cluster.
- $\text{Dis}(C1, C2) = \sum \text{dist}(P_i, P_j) / |C1| * |C2|$

Centroid linkage distance

- In centroid linkage distance, the distance between two clusters is defined as the distance between the centroids of the two clusters.
- The centroid of a cluster is the mean of all the points in the cluster.
- Mathematically, if we have two clusters A and B, and $c(A)$ and $c(B)$ are the centroids of clusters A and B, respectively, then the centroid linkage distance between A and B is given by:
$$d(A,B) = ||c(A) - c(B)||$$
where $||\cdot||$ denotes the Euclidean distance between the centroids.
- This distance metric tends to produce well-separated clusters that are roughly equally sized and shaped.
- It is often used in situations where the goal is to minimize the average distance between points in a cluster.
- However, it can be sensitive to outliers and may not work well when the clusters have different densities or sizes.

Ward's method

Aims to minimize the increase in variance (or sum of squared errors) when merging two clusters.

The distance between two clusters in Ward's method is calculated as follows:

1. First, calculate the centroid of each cluster (i.e., the mean of all the points in the cluster).
2. Then, merge the two clusters and calculate the new centroid of the merged cluster.
3. Finally, calculate the increase in variance (or sum of squared errors) caused by merging the two clusters.

This can be calculated using the following formula:

$$\Delta V = V(A \cup B) - V(A) - V(B)$$

where A and B are the two clusters being merged,

$A \cup B$ is the merged cluster, and

$V(\cdot)$ is the variance (or sum of squared errors) of the points in the cluster.

Note: In some implementations of Ward's method, the increase in variance is divided by the total number of points in the merged cluster.

- 4.
5. Repeat steps 2 and 3 for all possible pairs of clusters, and choose the pair that results in the smallest increase in variance (or sum of squared errors) as the clusters to merge next.
6. Continue merging clusters until there is only one cluster left (i.e., all points have been grouped into a single cluster).

Purpose of HC

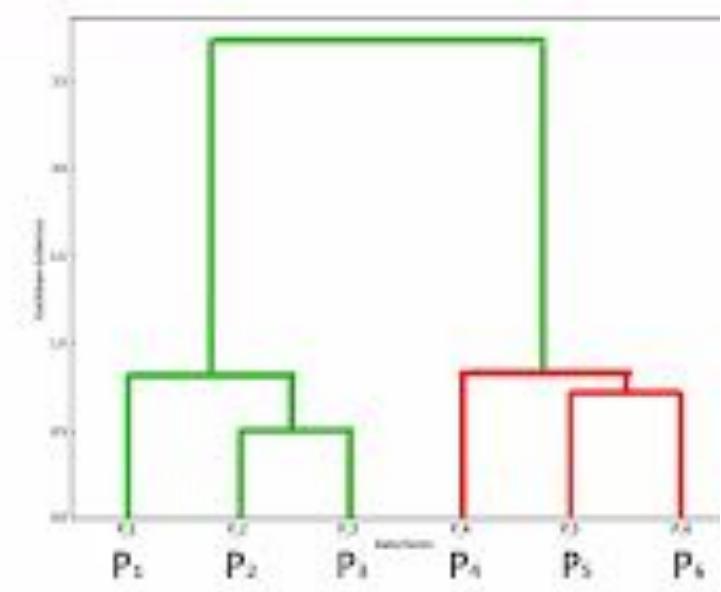
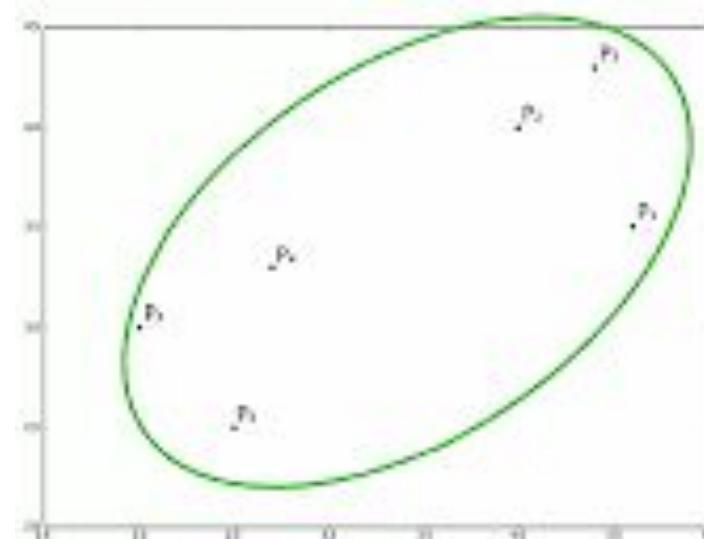
- How we get the actual result: right number of clusters?
- It maintains record of how these clusters are made during this process
 - record is stored in a dendrogram

Dendrogram

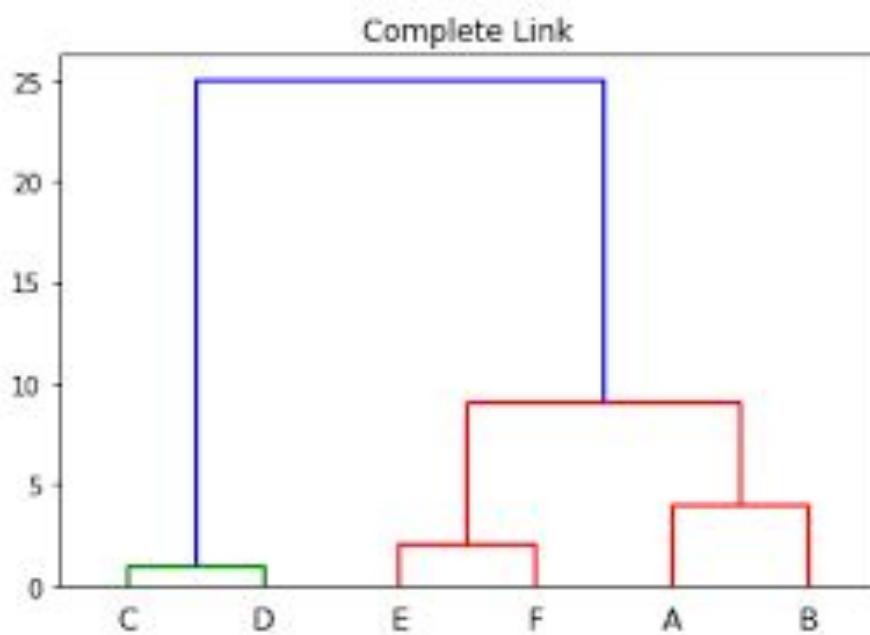
- A **Dendrogram** is a tree-like diagram that records the sequences of merges or splits.

How do dendograms work?

- 6 points on the left chart.
- Points are having dissimilarity
- We can measure this dissimilarity by finding distance between them.
- Using these we will create dendograms.
- Initially every single point is an individual cluster.

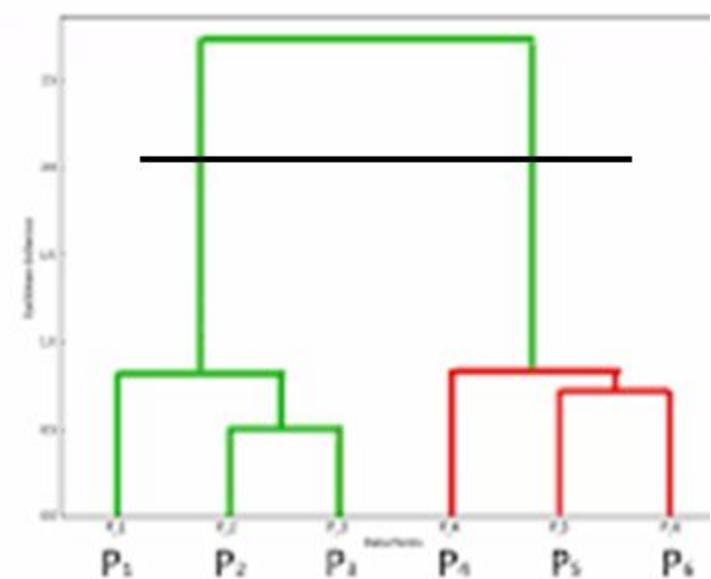


Another Dendrogram



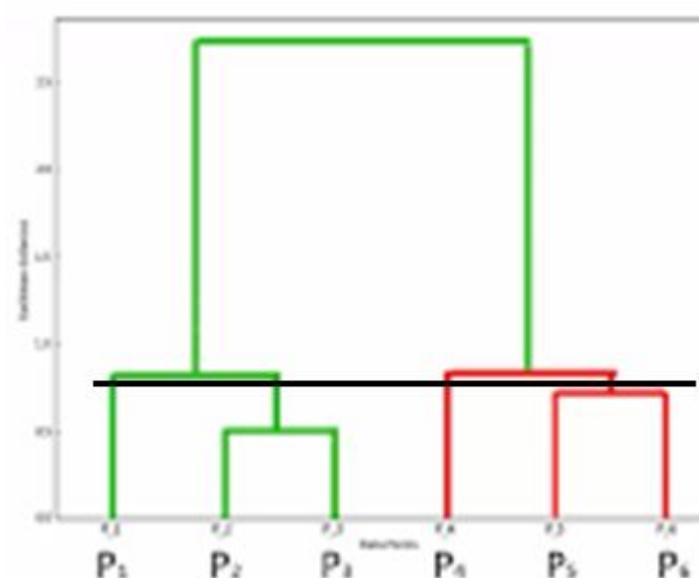
Using Dendograms

- Set a threshold for dissimilarity
- Not allowing any clusters that would have dissimilarity greater than that level (for eg. 1.7)
- Number of clusters
 - Number of lines the dissimilarity threshold line crosses.
- Here it suggest two clusters
 - P1, P2 and P3
 - P4, P5 and P6



Another example

- 4 clusters
 - P1
 - P2 and P3
 - P4
 - P5 and P6



Dendrogram suggest optimal number of clusters

- Standard approach
 - Find the longest vertical line (distance) that does not cross any of the horizontal line.
 - Take the threshold that cross this largest distance and use this threshold to find number of clusters.
- Any other method like elbow method can also be used.

Silhouette analysis

- to evaluate the quality of clustering results and
- to determine the optimal number of clusters in hierarchical clustering.
- The technique is based on the concept of silhouette coefficients, which measure
 - how well a data point fits into its assigned cluster compared to the other clusters.

Silhouette coefficient

- The silhouette coefficient for a data point is calculated as follows:
 1. Calculate the average distance between the data point and all other data points in the same cluster- $a(i)$.
 2. Calculate the average distance between the data point and all other data points in the nearest neighboring cluster (i.e., the cluster with the smallest average distance)- $b(i)$.
 3. Calculate the silhouette coefficient for the data point as $(b(i) - a(i)) / \max(a(i), b(i))$.

Silhouette coefficient

- The silhouette coefficient ranges from -1 to 1,
- A coefficient close to 1 indicates
 - data point is well-matched to its own cluster and poorly matched to neighboring clusters,
- A coefficient close to -1 indicates
 - Opposite
- A coefficient close to 0 indicates
 - the data point is close to the boundary between two clusters.

Finding optimal number of clusters using silhouette analysis

1. Perform hierarchical clustering on your data set for different values of k (the number of clusters).
2. For each value of k , calculate the average silhouette coefficient across all data points in the clustering result.
3. Plot the average silhouette coefficient as a function of k .
4. The optimal number of clusters is the value of k that maximizes the average silhouette coefficient.

Divisive Clustering

- is a top-down clustering approach
- Steps of Divisive Clustering
 1. Initially, all points in the dataset belong to one single cluster.
 2. Partition the cluster into two least similar cluster
 3. Proceed recursively to form new clusters until the desired number of clusters is obtained.

How to choose which cluster to split?

- Check the sum of squared errors of each cluster and choose the one with the largest value.
- use Ward's criterion to choose for the largest reduction in the difference in the SSE criterion as a result of the split

How to split the above-chosen cluster?

- how to split the chosen cluster into 2 clusters
 - Using k-means clustering
 - recursive k-means

Comparison

- Divisive algorithm is more accurate
 - Agglomerative clustering makes decisions by considering the local patterns or neighbor points without initially taking into account the global distribution of data.
 - These early decisions cannot be undone.
 - whereas divisive clustering takes into consideration the global distribution of data when making top-level partitioning decisions.
- Divisive clustering is more efficient if we do not generate a complete hierarchy all the way down to individual data leaves.
 - Time complexity of a naive agglomerative clustering is $O(n^3)$ because we exhaustively scan the $N \times N$ matrix `dist_mat` for the lowest distance in each of $N-1$ iterations.
 - Using priority queue data structure we can reduce this complexity to $O(n^2 \log n)$.
 - By using some more optimizations it can be brought down to $O(n^2)$.
 - Whereas for divisive clustering given a fixed number of top levels, using an efficient flat algorithm like K-Means, divisive algorithms are linear in the number of patterns and clusters.
- Divisive clustering is good at identifying large clusters while agglomerative clustering is good at identifying small clusters.

Dataset

CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40
6	Female	22	17	76
7	Female	35	18	6
8	Female	23	18	94
9	Male	64	19	3
10	Female	30	19	72
11	Male	67	19	14

Importing the libraries

- `import numpy as np`
- `import matplotlib.pyplot as plt`
- `import pandas as pd`

Importing the dataset

- dataset = pd.read_csv('Mall_Customers.csv')
- X = dataset.iloc[:, [3, 4]].values

Using the dendrogram to find the optimal number of clusters

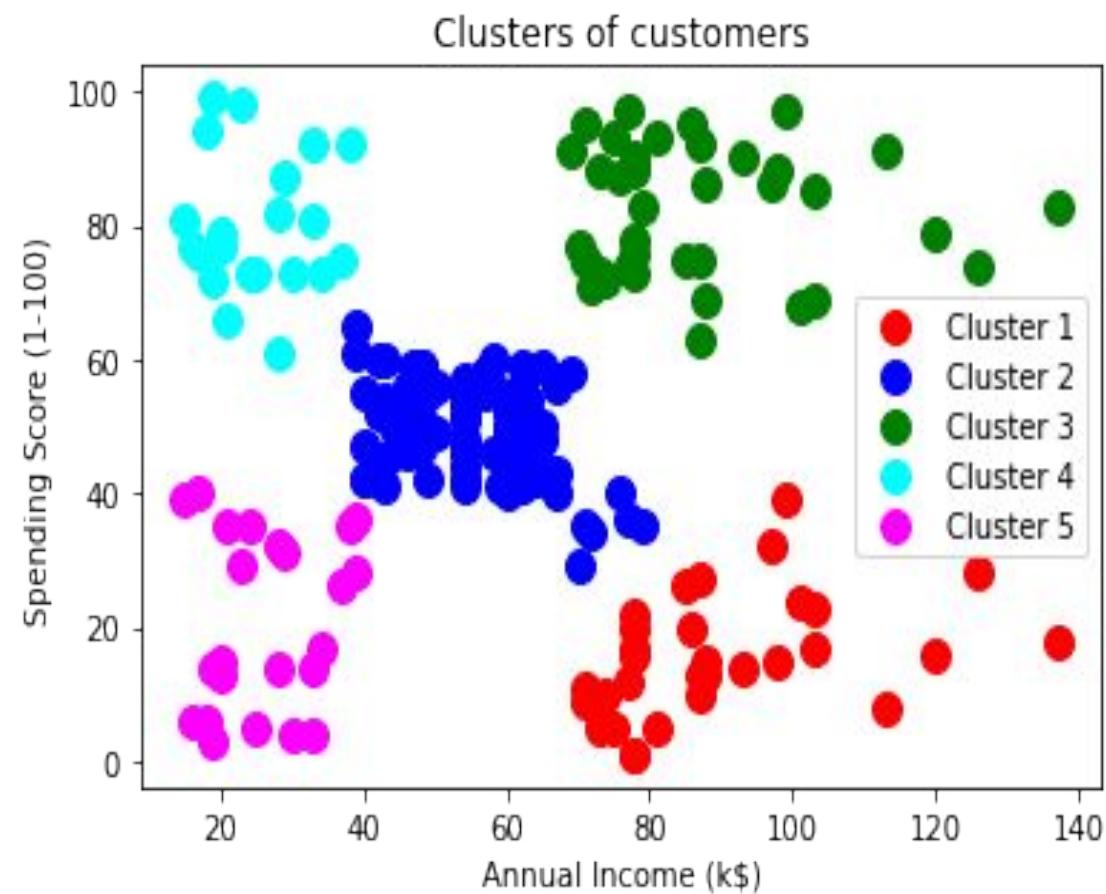
- `import scipy.cluster.hierarchy as sch`
- `dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))` // ward is minimum variance method
- `plt.title('Dendrogram')`
- `plt.xlabel('Customers')` // observation points
- `plt.ylabel('Euclidean distances')`
- `plt.show()`

Training the Hierarchical Clustering model on the dataset

- `from sklearn.cluster import AgglomerativeClustering`
- `hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')`
- `y_hc = hc.fit_predict(X)`
- `Print(y_hc)`

Visualising the clusters

```
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



Fuzzy C-Means Clustering

Fuzzy C-means (FCM)

- Fuzzy C-means (FCM) is a clustering algorithm that allows each data point to belong to multiple clusters with varying degrees of membership.
- Similar to K-means
 - it aims to partition a set of n-dimensional data points into a predefined number of clusters,
- Differs
 - assigns a membership grade for each cluster, rather than being assigned to a single cluster

Fuzziness Parameter (m)

- Controls how “fuzzy” the clusters are
- $m=1 \rightarrow$ behaves like hard clustering (k-means)
- $m>1 \rightarrow$ more fuzzy, overlapping clusters (usually 1.5–2.5)

Working of the FCM algorithm

1. Choose
 - the number of clusters (c) to form and
 - a fuzziness parameter (m) that determines the degree of fuzziness in the clustering.
2. Initialize the cluster centers randomly or by using some other method. OR
Initialize the membership matrix $U=[u_{ij}]$ randomly such that $\sum (j=1 \text{ to } c) u_{ij}=1$ for all i
1. Calculate the membership grade of each point for each cluster using the following formula:
 - $u_{ij} = (1 / \text{sum}(k=1 \text{ to } c) [(d_{ij} / d_{ik}) ^ {(2 / (m-1))}])$
 - where u_{ij} is the membership grade of data point i for cluster j ,
 - d_{ij} is the distance between data point i and cluster center j , and
 - d_{ik} is the distance between data point i and cluster center k .

OR Step 4...

Working of the FCM algorithm

4. Update the cluster centers using the membership grades of the data points as follows:

- $v_j = (\sum_{i=1}^n (u_{ij}^m * x_i)) / (\sum_{i=1}^n (u_{ij}^m))$
- where v_j is the new center of cluster j,
- x_i is data point i, and
- u_{ij} is the membership grade of data point i for cluster j.

OR

Step 3...

5. Repeat steps 3-4 until the cluster centers (or membership values) change very little.

6. Assign each data point to the cluster with the highest membership grade.

Example

- Suppose we have a dataset of 10 two-dimensional points:
- $[(1, 3), (2, 3), (3, 3), (4, 3), (5, 3), (2, 2), (3, 2), (4, 2), (3, 1), (4, 1)]$
- We want to cluster this dataset into 2 clusters using FCM. Let's set the fuzziness parameter m to 2 and the number of iterations to 10.
- We'll initialize the cluster centers randomly, and let's assume they are:
 - $[(2, 2), (4, 2)]$

Working of FCM algorithm on this dataset

1. Calculate the Euclidean distances between each data point and each cluster center.
2. Calculate the membership grades of each data point for each cluster using the formula:
 - Memberships: [[0.528, 0.472], [0.542, 0.458], [0.553, 0.447], [0.574, 0.426], [0.596, 0.404], [0.437, 0.563], [0.425, 0.575], [0.414, 0.586], [0.545, 0.455], [0.537, 0.463]]
3. Calculate the new cluster centers using the membership grades of the data points as follows:
 - New centers: [(2.67, 2.17), (3.45, 2.44)]

4. Repeat steps 2-3 for a total of 10 iterations or until the membership grades converge to a stable value.

After 10 iterations, the membership grades and cluster centers converge to the following values:

Memberships: [[0.756, 0.244], [0.732, 0.268], [0.709, 0.291],
[0.681, 0.319], [0.653, 0.347], [0.305, 0.695], [0.284, 0.716],
[0.263, 0.737], [0.39, 0.61], [0.37, 0.63]]

New centers: [(3.1, 2.7), (3.8, 1.9)]

5. Assign each data point to the cluster with the highest membership grade.

Advantages

- Handles **uncertain or overlapping data** better than k-means.
- Produces **soft partitions**, useful when boundaries between clusters are not well-defined.

Disadvantages

- Requires choosing the number of clusters c and fuzziness m .
- Sensitive to **initialization** and **local minima**.
- **Computationally more expensive** than k-means.