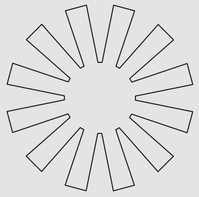
IET

ML BOOTCAMP

2024

TABLE OF CONTENTS

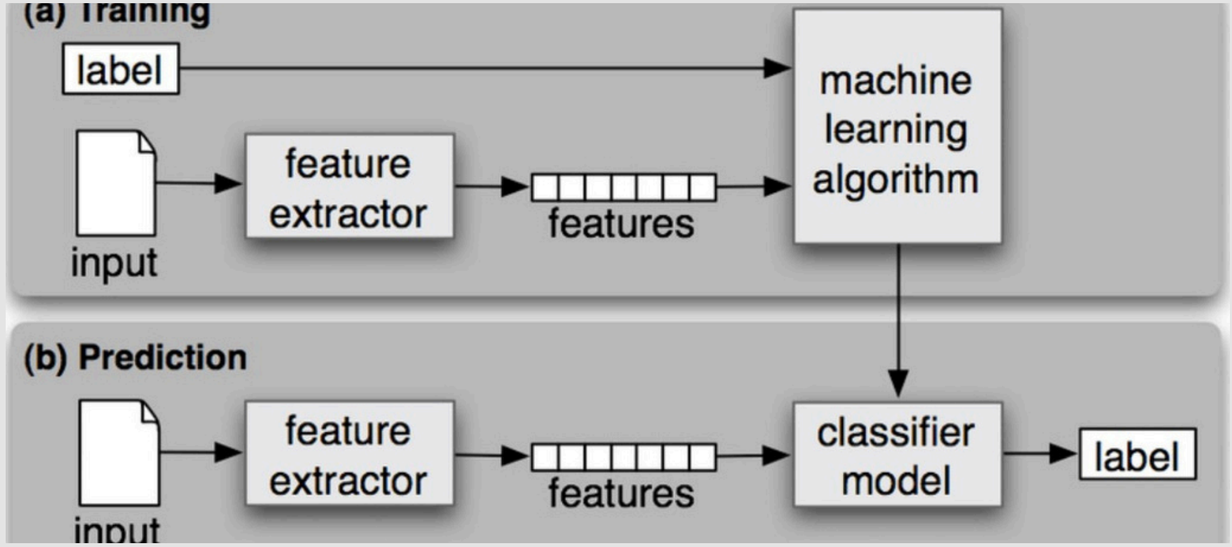


MACHINE LEARNING

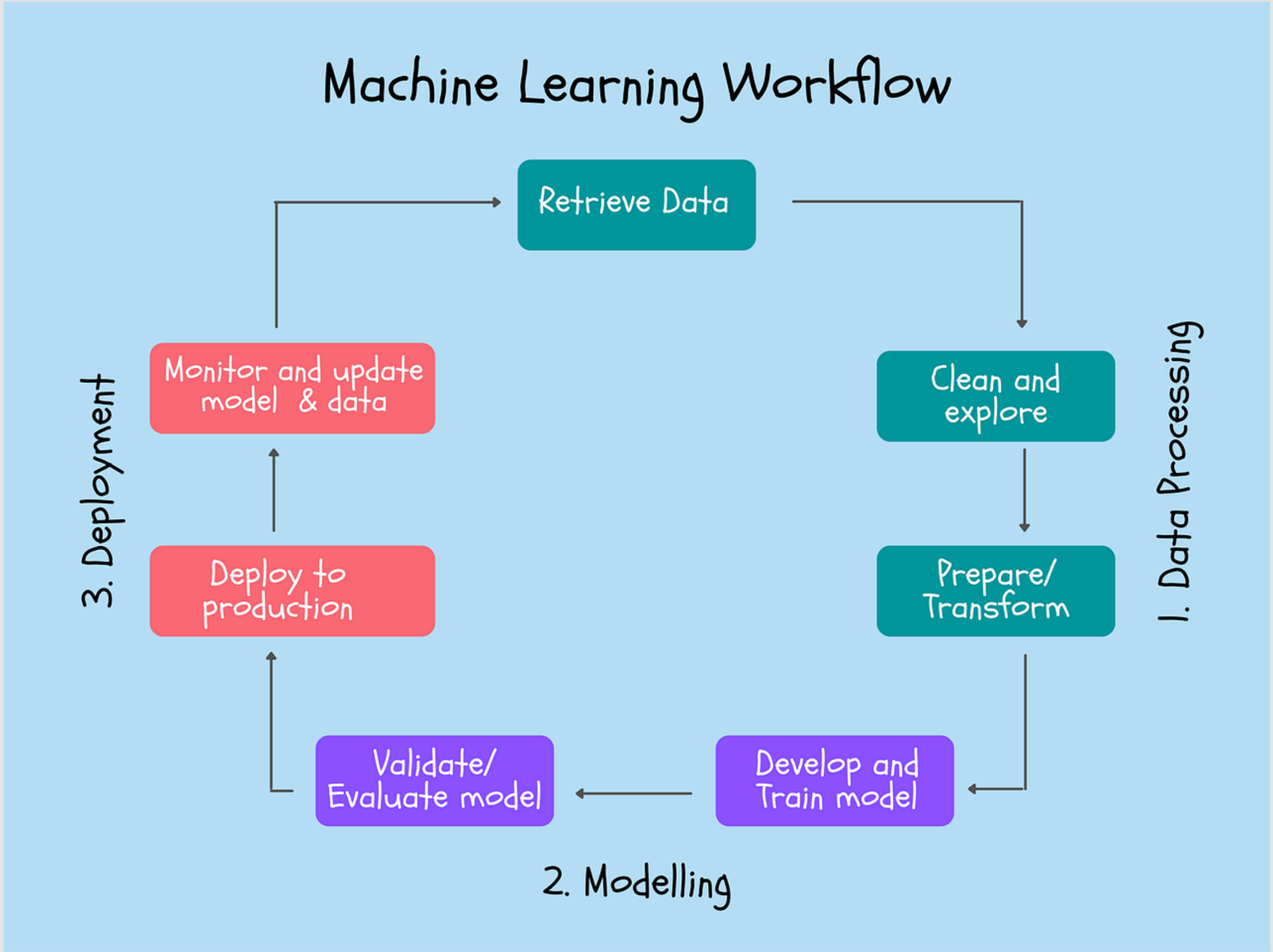
WHAT:
"MACHINE LEARNING IS THE FIELD OF STUDY THAT GIVES COMPUTERS THE ABILITY TO LEARN FROM DATA WITHOUT BEING EXPLICITLY PROGRAMMED."

HOW:
LEARNING PROCESS:
DATA → MODEL TRAINING → PREDICTION

WHY:
REAL-WORLD EXAMPLES:
SPAM DETECTION, SELF-DRIVING CARS,RECOMMENDATION SYSTEMS



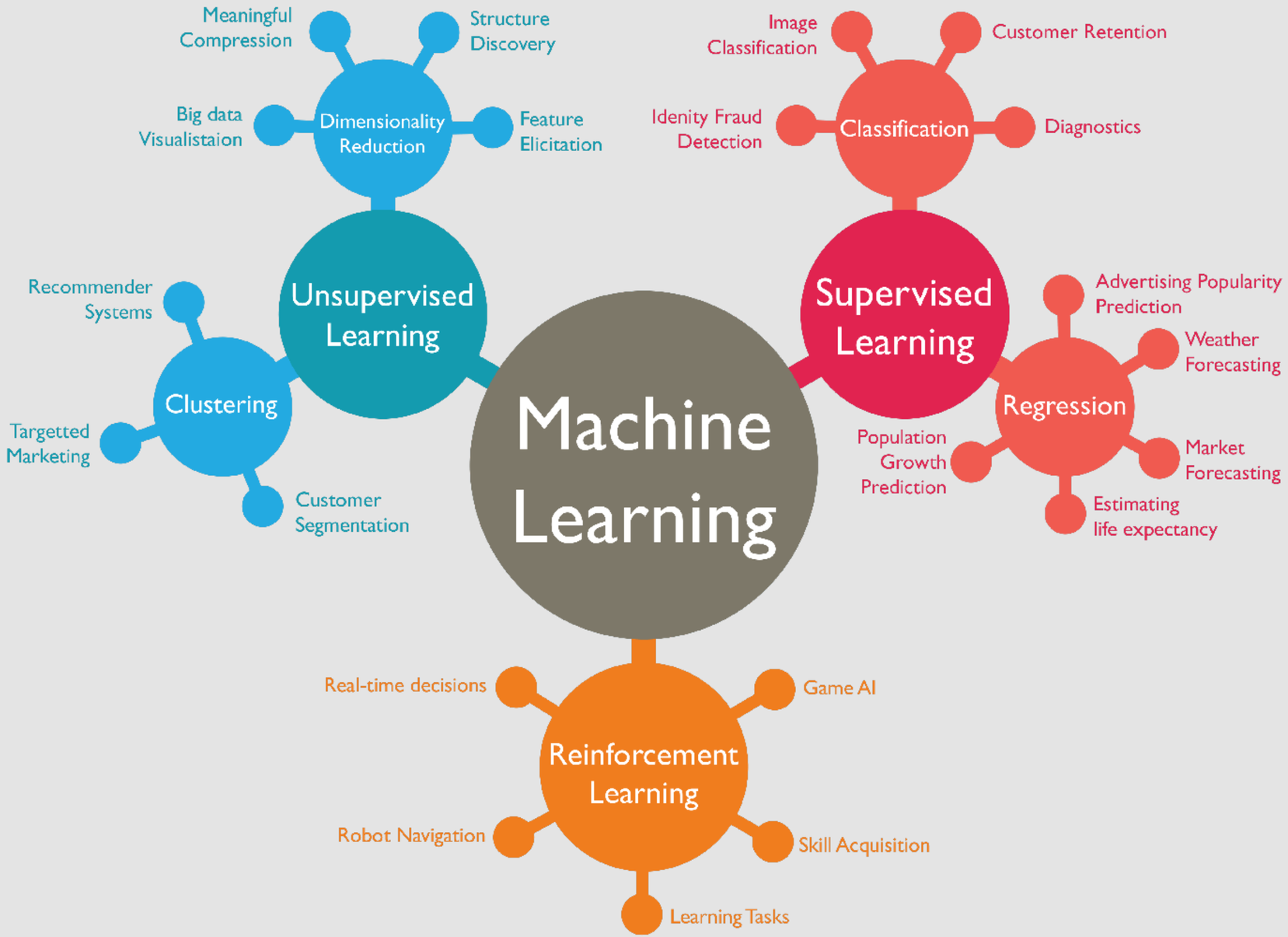
GENERAL WORKFLOW



TYPES OF MACHINE LEARNING

Type	What	When to Use	Example
Supervised learning	Supervised learning uses labeled data to train models to predict outputs based on inputs.	When you have historical data with known outcomes.(e.g., regression, classification)	Predicting house prices based on size and location.
Unsupervised learning	Unsupervised learning finds hidden patterns or intrinsic structures in unlabeled data.	Use when the data lacks labeled outputs, and you want to discover groupings or patterns (e.g., clustering, anomaly detection).	Customer segmentation using K-Means Clustering to group similar customers for targeted marketing.
Reinforcement Learning	Reinforcement learning trains an agent to make decisions by rewarding or penalizing its actions based on interactions with the environment.	Use when the goal is to learn optimal actions over time through trial and error in dynamic environments.	Training a robot to navigate a maze or AlphaGo learning to play Go by receiving rewards for winning moves.

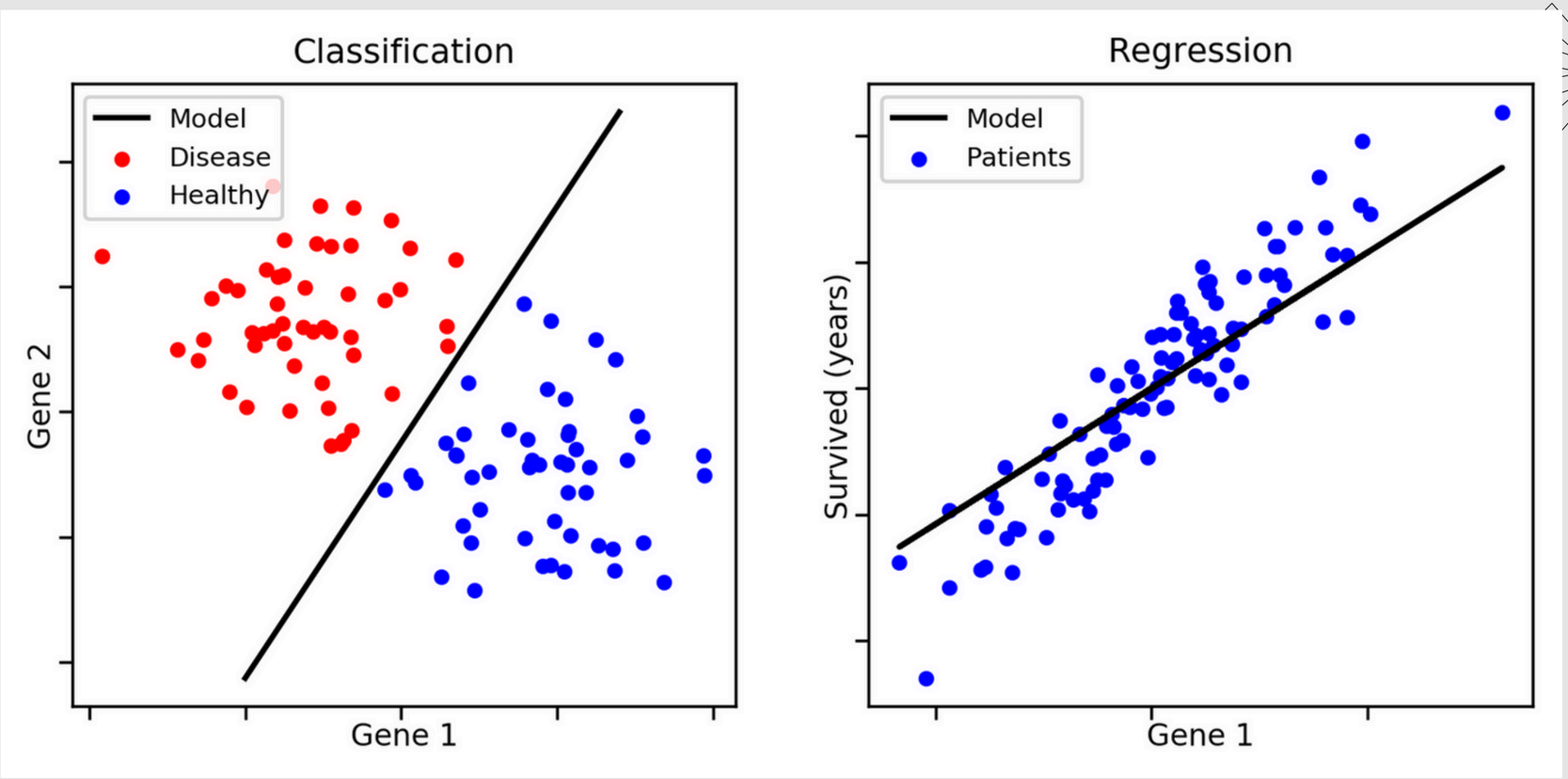
APPLICATIONS



SUPERVISED LEARNING

REGRESSION: PREDICTING CONTINUOUS OUTCOMES (E.G., STOCK PRICES, SALES).EXAMPLE: HOUSE PRICE PREDICTION (PRICE VS. SIZE)

CLASSIFICATION: PREDICTING CATEGORICAL OUTCOMES (E.G., CLASSIFYING IMAGES AS CAT OR DOG).EXAMPLE: EMAIL SPAM CLASSIFICATION



KEY TERMS FOR REGRESSION

- Regression is a type of supervised learning used to predict continuous outcomes based on input features. It models the relationship between the independent variables (inputs) and the dependent variable (output).

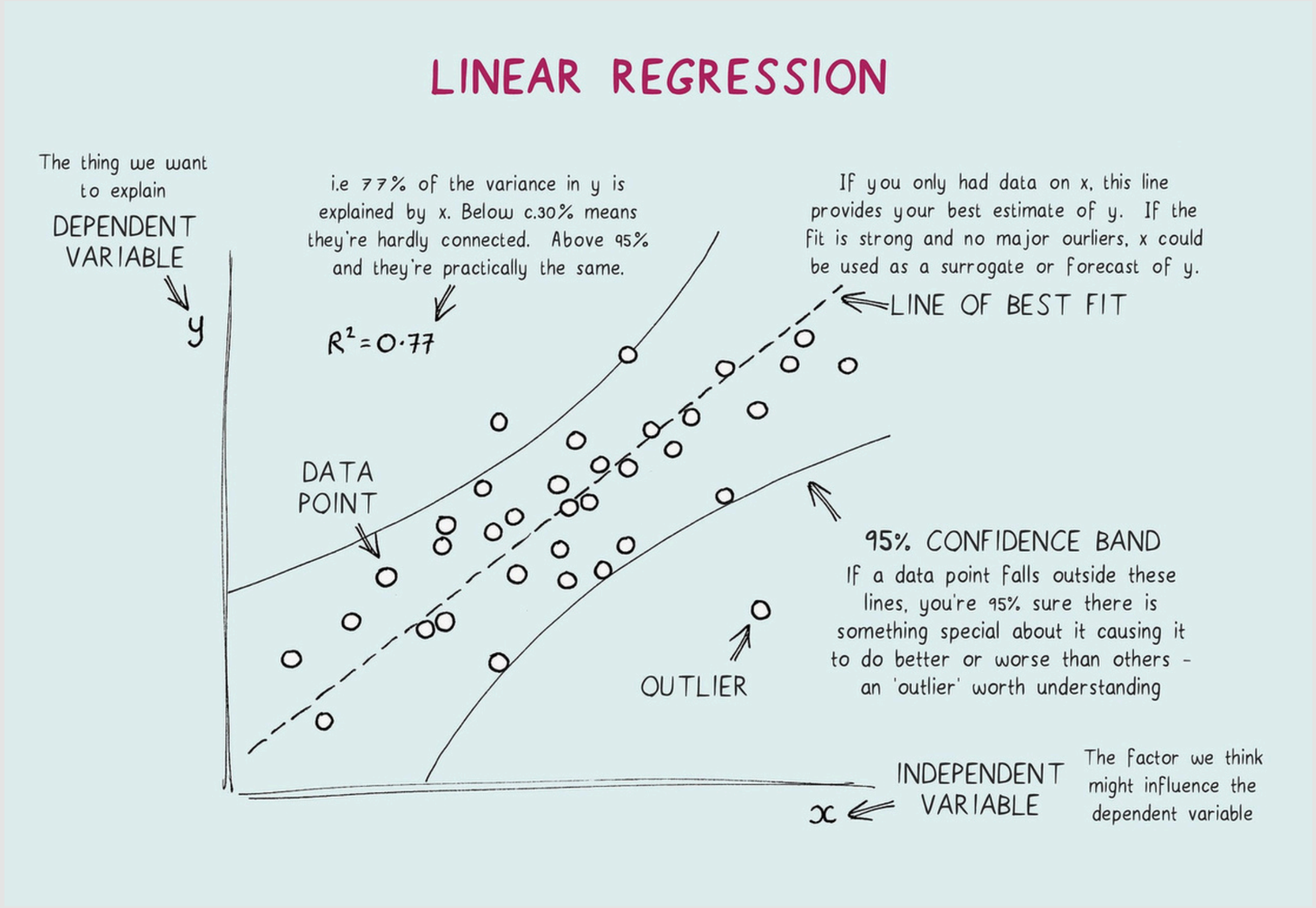
- Cost Function: In regression, the cost function measures the error between the predicted and actual values.

- Accuracy: The proportion of correct predictions out of the total predictions made.
- Precision: The ratio of correctly predicted positive observations to the total predicted positives (relevant in imbalanced datasets).
- Recall: The ratio of correctly predicted positives to all actual positives (useful in identifying all positive instances).
- F1 Score: The harmonic mean of precision and recall, balancing both for a more comprehensive evaluation of model performance.

MOST USED COST FUNCTIONS

Cost Function	Used in	Logic	Formula
Mean Squared Error (MSE)	Linear regression and other regression models.	Measures the average of the squared differences between the predicted and actual values. Minimizing MSE leads to better predictions.	$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
Mean Absolute Error (MAE)	Regression tasks	Measures the average of the absolute differences between predicted and actual values, which is less sensitive to outliers than MSE	$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$
Cross-Entropy Loss (Log Loss)	Classification problems, especially for Logistic Regression and neural networks.	Measures the difference between the actual class and the predicted probabilities, penalizing incorrect confident predictions heavily.	$\text{Cross-Entropy Loss} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$
Hinge Loss	Support Vector Machines (SVM) for classification tasks.	Ensures that the correct class has a score higher than the incorrect classes by a margin. The hinge loss function focuses on maximizing the margin between classes	$\text{Hinge Loss} = \max(0, 1 - y_i \cdot \hat{y}_i)$

LINEAR REGRESSION



- LINEAR REGRESSION MODELS THE RELATIONSHIP BETWEEN A DEPENDENT VARIABLE AND ONE OR MORE INDEPENDENT VARIABLES BY FITTING A STRAIGHT LINE.

LINEAR REGRESSION : FORMULA

For a linear relationship between the independent variables (features) and the dependent variable (output), the hypothesis function is expressed as:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Where:

- y^ is the predicted value.
- θ0 is the intercept (bias term).
- θ1,θ2,…,θn are the coefficients (weights) of the features.
- x1,x2,…,xn are the independent variables (features).

Vectorized Form:
For simplicity, this can be written in matrix form as:

$$\hat{y} = \mathbf{X}\theta$$

Where:

- y^ is the predicted value Column
- X is the feature matrix (including a column of 1’s for the intercept term).
- θ is the vector of parameters (including θ0\theta_0θ0, the bias term).

LINEAR REGRESSION : COST FUNCTION

Cost Function (Mean Squared Error):

The goal of linear regression is to minimize the cost function, typically Mean Squared Error (MSE):

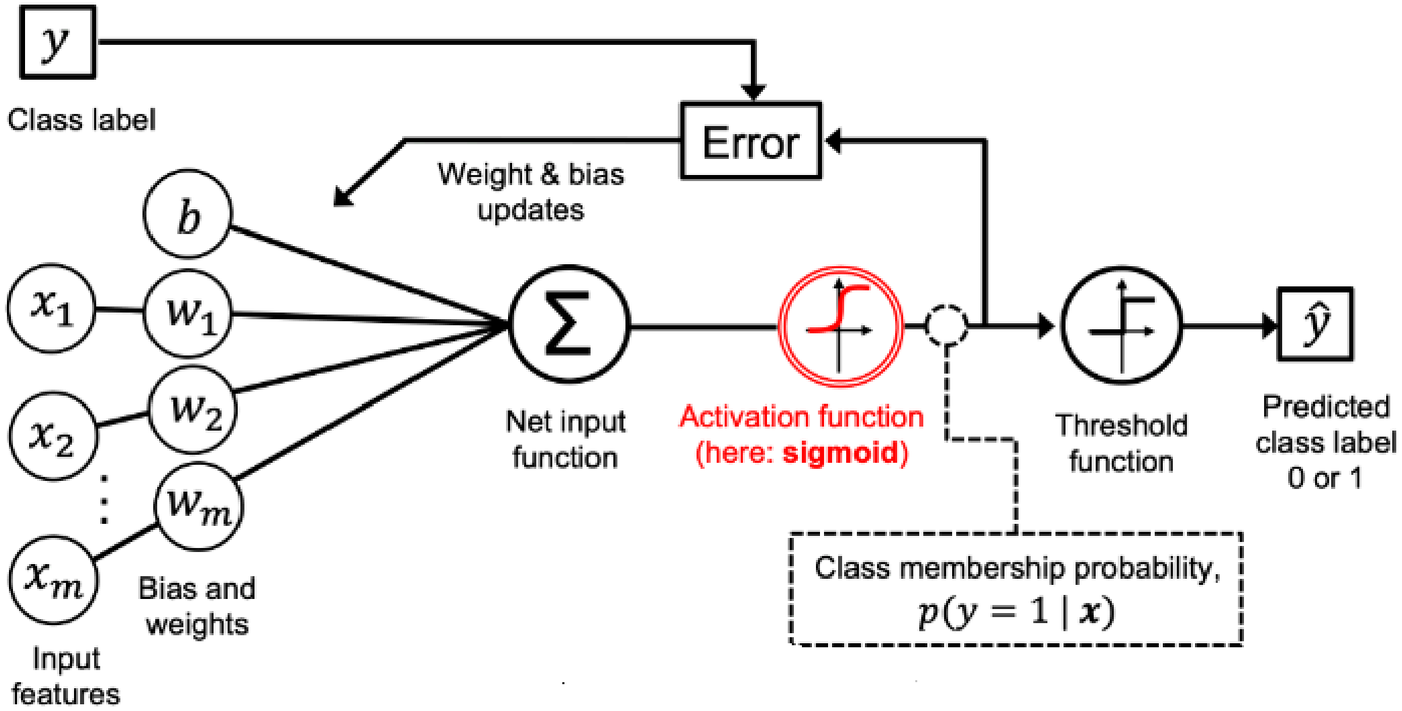
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$

WHERE:

- m is the number of training examples.
- y^(i) is the predicted value for the iii-th example.
- y(i) is the actual value for the iii-th example.

The model learns the parameters θ by minimizing this cost function using techniques like Gradient Descent.

LOGISTIC REGRESSION



LOGISTIC REGRESSION IS A SUPERVISED MACHINE LEARNING ALGORITHM USED FOR CLASSIFICATION TASKS WHERE THE GOAL IS TO PREDICT THE PROBABILITY THAT AN INSTANCE BELONGS TO A GIVEN CLASS OR NOT.

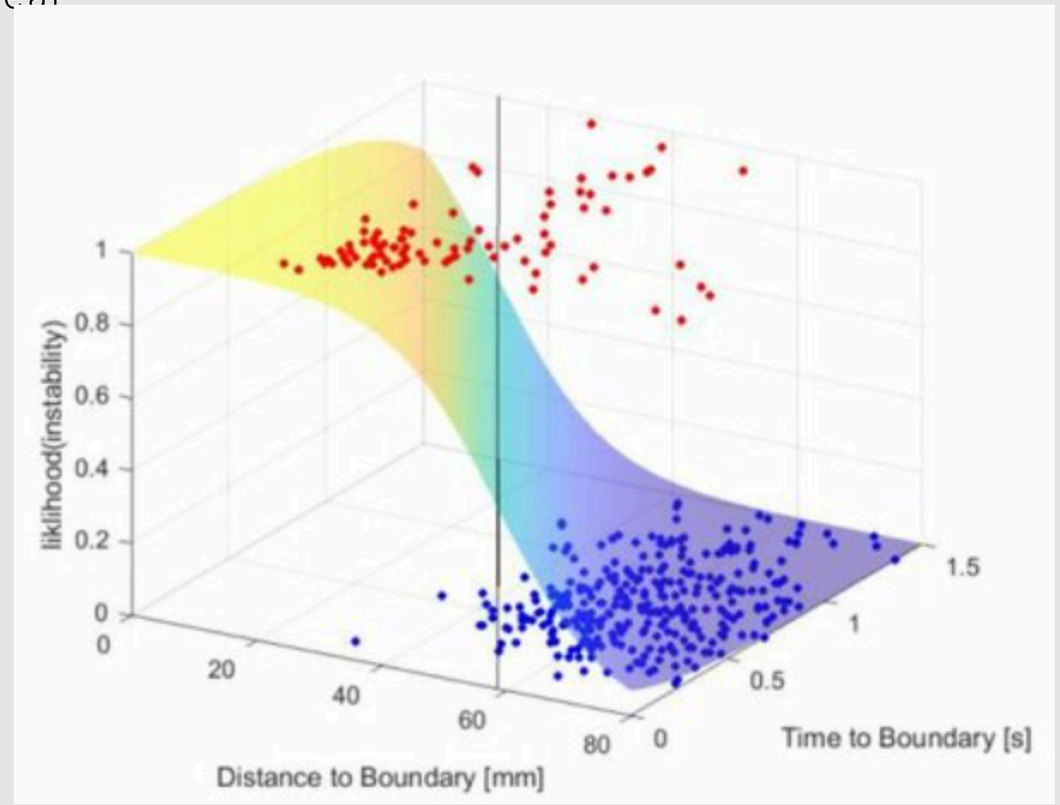
LOGISTIC REGRESSION IS A STATISTICAL ALGORITHM WHICH ANALYZE THE RELATIONSHIP BETWEEN TWO DATA FACTORS.

LOGISTIC REGRESSION

The assumption include:

1. Independent observations: Each observation is independent of the other. meaning there is no correlation between any input variables.
2. Binary dependent variables: It takes the assumption that the dependent variable must be binary or dichotomous, meaning it can take only two values. For more than two categories SoftMax functions are used.
3. Linearity relationship between independent variables and log odds: The relationship between the independent variables and the log odds of the dependent variable should be linear
4. No outliers: There should be no outliers in the dataset.
5. Large sample size: The sample size is sufficiently large

Logistic Regression can be classified into three types:
Binomial: In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
Multinomial: In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as “cat”, “dogs”, or “sheep”
Ordinal: In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as “low”, “Medium”, or “High”



LOGISTIC REGRESSION: WORKING

The logistic regression model transforms the linear regression function continuous value output into categorical value output using a sigmoid function, which maps any real-valued set of independent variables input into a value between 0 and 1. This function is known as the logistic function.

Let the independent input features be:

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ x_{21} & \cdots & x_{2m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}$$

and the dependent variable is Y having only binary value i.e. 0 or 1.

$$Y = \begin{cases} 0 & \text{if } Class\ 1 \\ 1 & \text{if } Class\ 2 \end{cases}$$

Then, apply the multi-linear function to the input variables X.

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

Here xi is the ith observation of X, wi=[w1,w2,w3,⋯,wm]wi=[w1,w2,w3,⋯,wm] is the weights or Coefficient, and b is the bias term also known as intercept. simply this can be represented as the dot product of weight and bias.

$$z = w \cdot X + b$$

SIGMOID AND SOFTMAX FUNCTION

Sigmoid function converts the continuous variable data into the probability i.e. between 0 and 1.

- $\sigma(z)$ tends towards 1 as $z \to \infty$
- $\sigma(z)$ tends towards 0 as $z \to -\infty$
- $\sigma(z)$ is always bounded between 0 and 1

  where the probability of being a class can be measured as:

  $P(y=1)=\sigma(z)$

  $P(y=0)=1-\sigma(z)$

The softmax function, on the other hand, is used for multi-class classification, normalizing the output into a probability distribution across multiple classes, ensuring that the sum of the probabilities is 1.

$$\text{Softmax}\,\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$\text{Sigmoid}\,S(x) = \frac{1}{1+e^{-x}}$$

CROSS-ENTROPY LOSS FUNCTION

While the sigmoid function gives the prediction value, the cross-entropy function is used to evaluate model learning capabilities and enhance them.
The cross-entropy (log) loss function for logistic regression measures the difference between the predicted probabilities and the actual binary class labels (0 or 1). It is defined as:

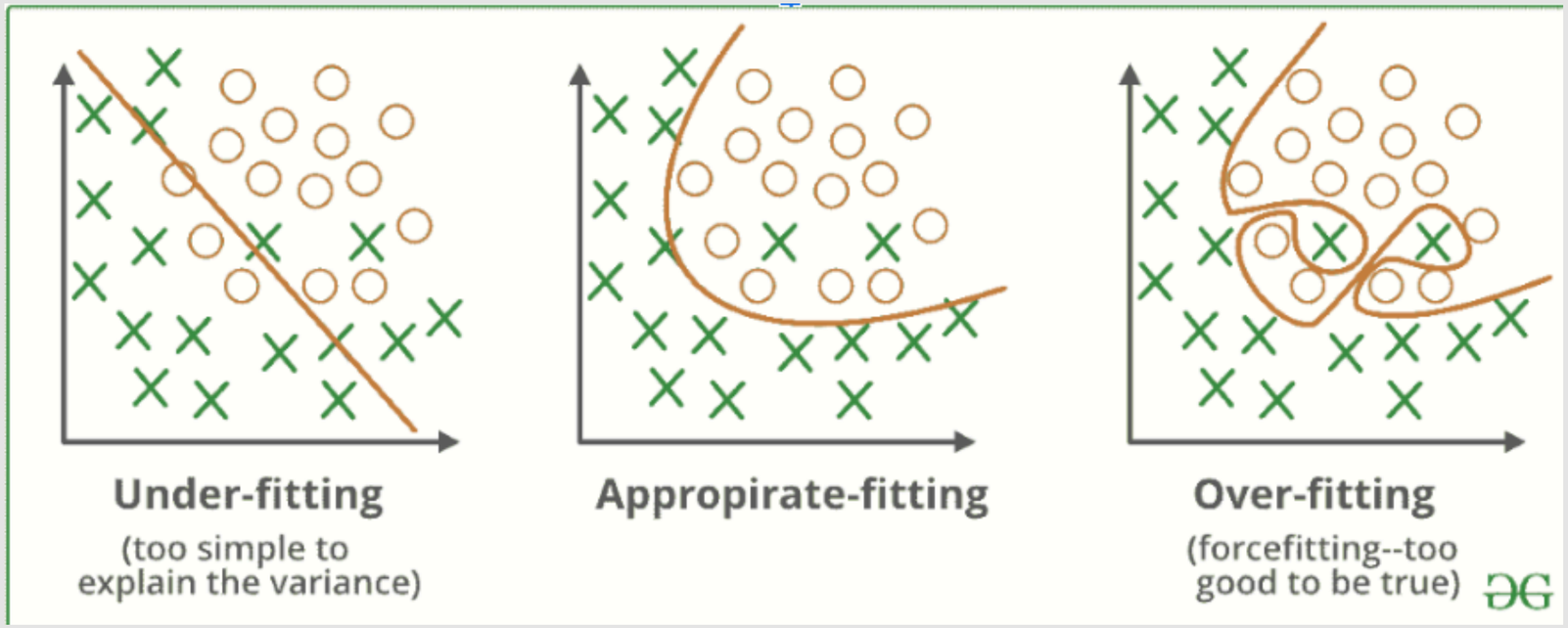
$$\text{Log Loss} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

where:

- yi is the actual label (0 or 1),
- pi is the predicted probability of the label being 1,
- n is the number of samples.

  The log loss penalizes wrong predictions more heavily as the predicted probability diverges from the actual class, making it a suitable cost function for logistic regression. The goal is to minimize this loss during training to improve model accuracy.

UNDERFITTING AND OVERFITTING



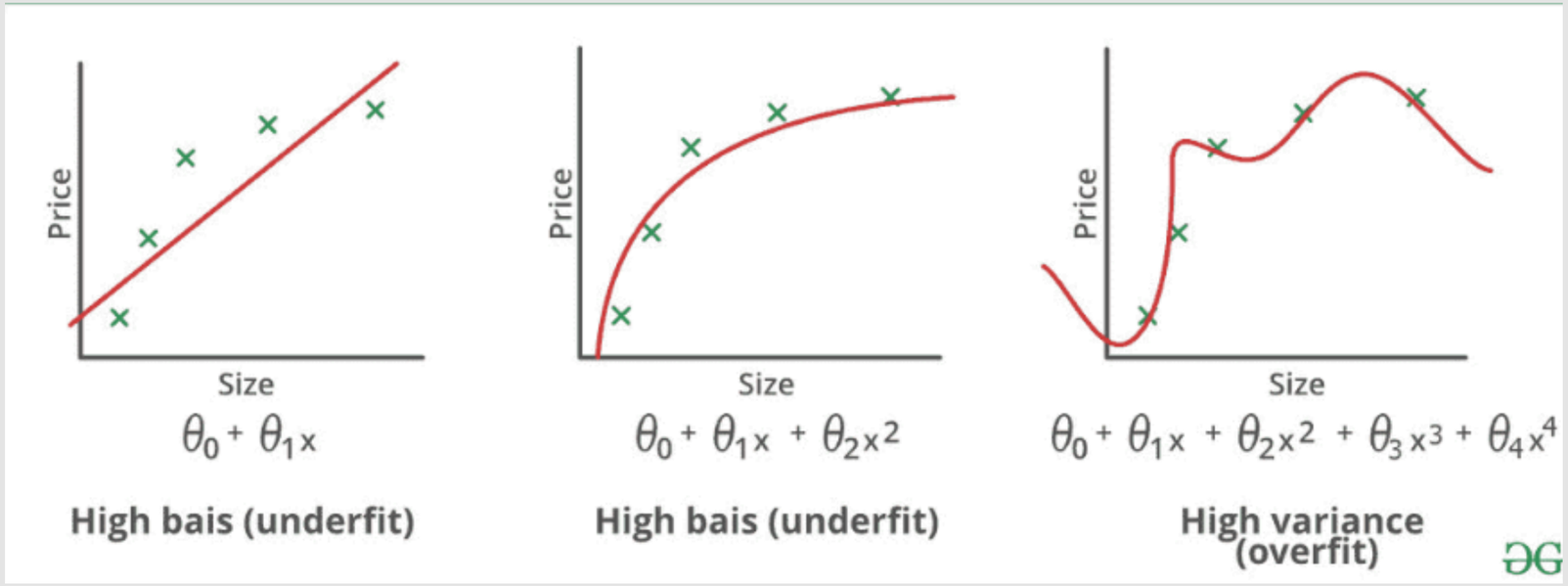
Overfitting is a phenomenon that occurs when a Machine Learning model is constrained to the training set and not able to perform well on unseen data. That is when our model learns the noise in the training data as well. This is the case when our model memorizes the training data instead of learning the patterns in it.

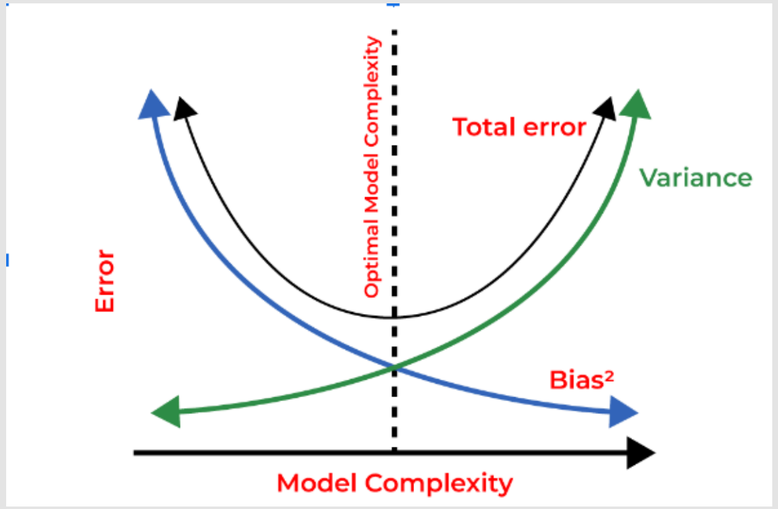
Underfitting on the other hand is the case when our model is not able to learn even the basic patterns available in the dataset. In the case of the underfitting model is unable to perform well even on the training data hence we cannot expect it to perform well on the validation data. This is the case when we are supposed to increase the complexity of the model or add more features to the feature set

BIAS AND VARIANCE TRADEOFF

The bias-variance tradeoff in machine learning describes the balance between two types of errors that affect a model's performance:

Bias refers to error due to overly simplistic models that make strong assumptions, leading to underfitting. High bias results in poor performance on both training and test data because the model fails to capture the true patterns in the data.	Variance refers to error due to overly complex models that are sensitive to small fluctuations in the training data, leading to overfitting. High variance results in good performance on the training data but poor generalization to new data.





The tradeoff is that reducing bias typically increases variance and vice versa. The goal is to find the right balance for optimal model performance.

REGULARIZATION

While developing machine learning models you must have encountered a situation in which the training accuracy of the model is high but the validation accuracy or the testing accuracy is low.

Regularization introduces a penalty for more complex models, effectively reducing their complexity and encouraging the model to learn more generalized patterns. This method strikes a balance between underfitting and overfitting, where underfitting occurs when the model is too simple to capture the underlying trends in the data, leading to both training and validation accuracy being low.

Lasso (Least Absolute Shrinkage and Selection Operator) adds a penalty equal to the absolute value of the coefficients (L1 norm) to the loss function, which can shrink some coefficients to zero, effectively performing feature selection.

$$\text{cost} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{m} |w_i|$$

Ridge regularization adds a penalty equal to the square of the coefficients (L2 norm), which shrinks the coefficients but does not zero them out, leading to smaller, non-zero values for all features, helping reduce model complexity and multicollinearity.

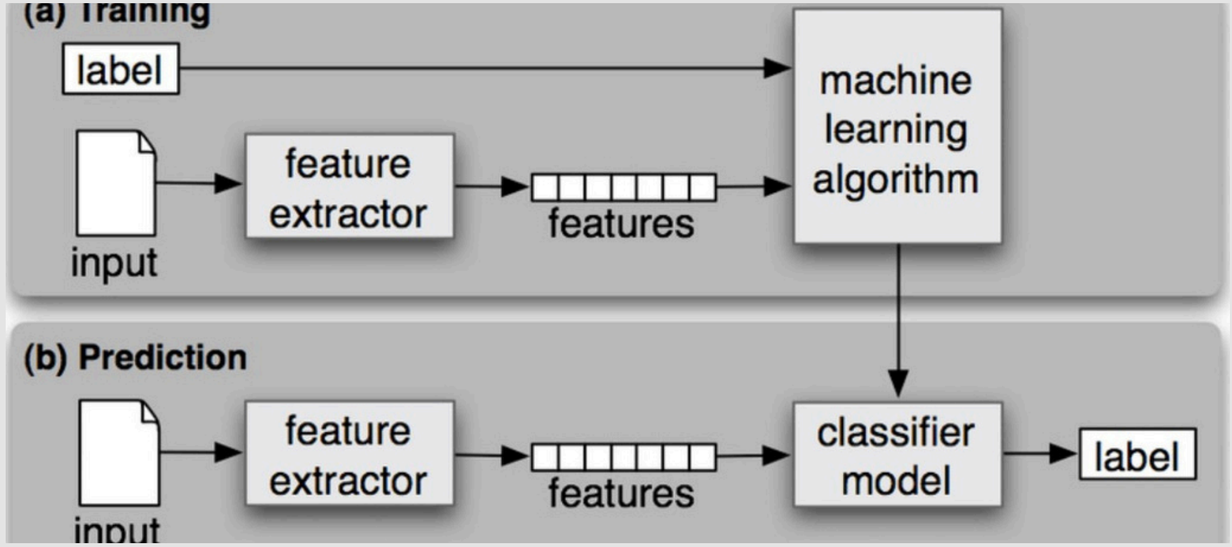
$$\text{cost} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{i=1}^{m} w_i^2$$

MACHINE LEARNING

WHAT:
"MACHINE LEARNING IS THE FIELD OF STUDY THAT GIVES COMPUTERS THE ABILITY TO LEARN FROM DATA WITHOUT BEING EXPLICITLY PROGRAMMED."

HOW:
LEARNING PROCESS:
DATA → MODEL TRAINING → PREDICTION

WHY:
REAL-WORLD EXAMPLES:
SPAM DETECTION, SELF-DRIVING CARS,RECOMMENDATION SYSTEMS



NAIVE BAYES ALGORITHM

- The naive Bayes algorithm is purely probabilistic. This means that the output is a number between 0 to 1, representing the probability of the label begin True(possible labels are True and False)
- The main component of naive Bayes is Bayes Theorem.
- Bayes Theorem is a fundamental theorem in probability and statistics, and it helps us calculate probabilities. The more we know about a particular situation, the more accurate the probability is.
- However, applying the Bayes theorem becomes complex when the amount of information becomes large. Hence, an assumption is made to simplify the process. The assumption is not necessarily true, but that works pretty well in practice and simplifies our lives quite a lot. This simplified algorithm is called as Naive Byes Algorithm
- There are several types of Naive Bayes algorithms, but the most commonly used version is called Multinomial Naive Bayes.

NAIVE BAYES ALGORITHM

Take an example:

- let’s say we want to find the probability that it will snow today.
- If we have no information about where we are and what time of the year it is, we can only come up with a vague estimate.
- If we are given information that we are in Norway, then the probability that it will snow today increases.
- If we are given information that we are in Mangalore, this probability would decrease drastically.
- Bayes theorem helps us calculate these new probabilities once we know more about the situation.
- However, when we have a lot of information, the math becomes complicated.
- If I were to ask you the probability that it will snow today given the following: we are in Norway, it is February, the temperature is -10 degrees Celsius, and the humidity is 30%
- then you would assume that the probability of snow is quite high, but exactly how high?
- Bayes theorem has a hard time dealing with so much information.

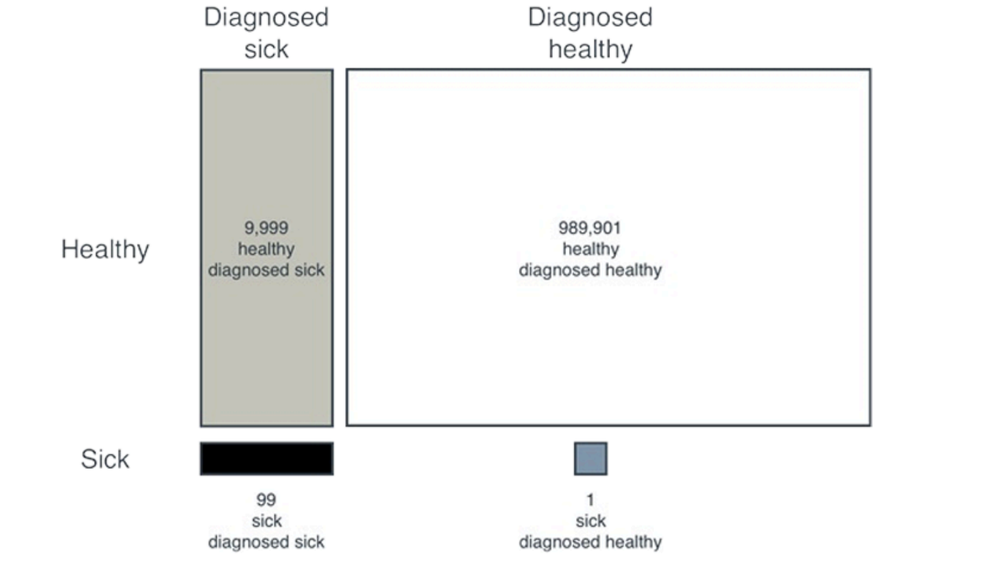
NAIVE BAYES ALGORITHM

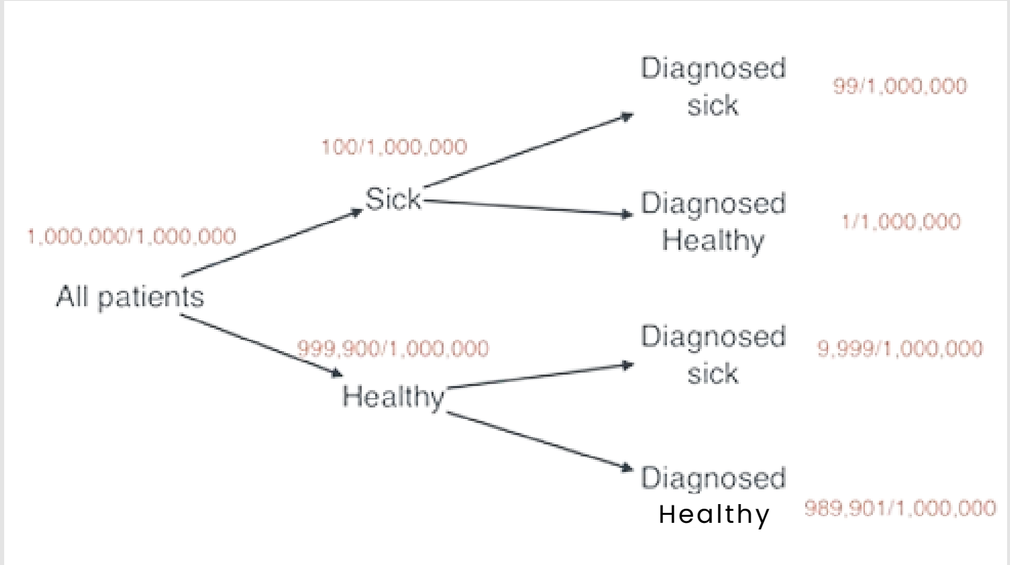
Let's take another example:

- Suppose there is a rare disease
- On average, 1 out of every 10,000 people have the disease
- There is a test for this disease that is 99% accurate
- What is the probability that someone has the disease?
- What is the probability that someone who has tested positive actually has the disease?

NAIVE BAYES ALGORITHM

- Let’s take a large group of people, say, 1 million people.
- Out of 1 million people, 999,900 of them are healthy, and 100 of them are sick.
- The test is correct 99% of the time, meaning 99% correct in both the categories of sick and healthy.
- From the diagram, it is clear that the probability of being sick when diagnosed sick = 99/(99+9999) = 0.0098





NAIVE BAYES ALGORITHM

- Defining some terms:

The primary objective of Bayes' theorem is to compute a probability. Initially, without any information, we can only determine an initial probability known as the prior. Once an event occurs, it provides us with new information. With this information, we can refine our estimate of the probability we wish to calculate. This improved estimate is referred to as the posterior.

PRIOR - The initial probability that we calculate.

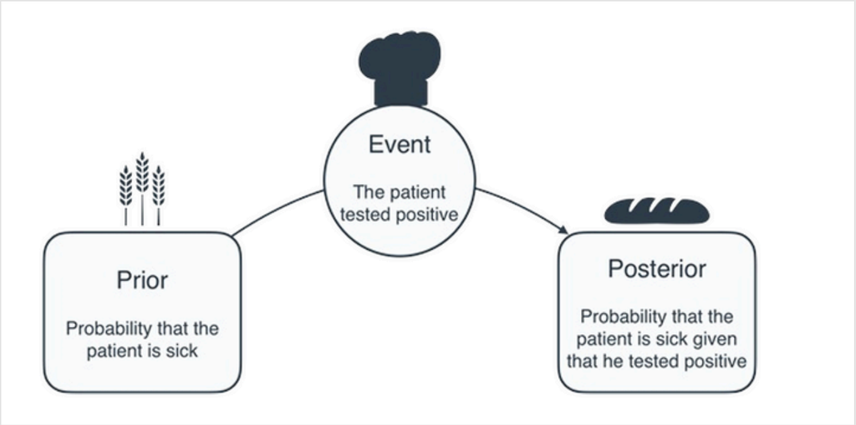
EVENT- What gives us information to calculate better probabilities.

POSTERIOR - The final (and more accurate) probability that we calculate using the prior probability and the event.

NAIVE BAYES ALGORITHM

- In the earlier example:

1. At first, the probability stands at 1/10,000, as our only knowledge is that one out of every 10,000 patients is unwell. Thus, this 1/10,000, or 0.0001, represents the prior probability.
2. Suddenly, new information emerges; in this scenario, the patient underwent a test and received a positive result.
3. Following this positive result, we recalibrate the probability of the patient being sick, which then calculates to 0.0098. This is known as the posterior probability.

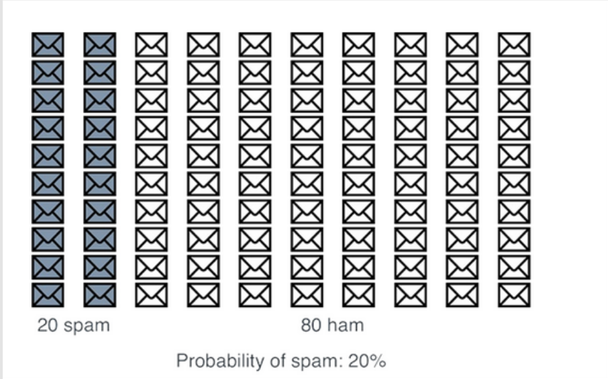


NAIVE BAYES ALGORITHM

- Example 3:

What is the probability that an email is spam? Given a new email, we want to determine the probability that it is either spam or legitimate (ham). This approach allows us to route emails with the highest chances of being spam straight to the spam folder while retaining the others in our inbox. This probability should be influenced by details related to the new email, such as the words used, the sender, the size, and more. For instance, an email containing the word 'sale' is generally more likely to be classified as spam.

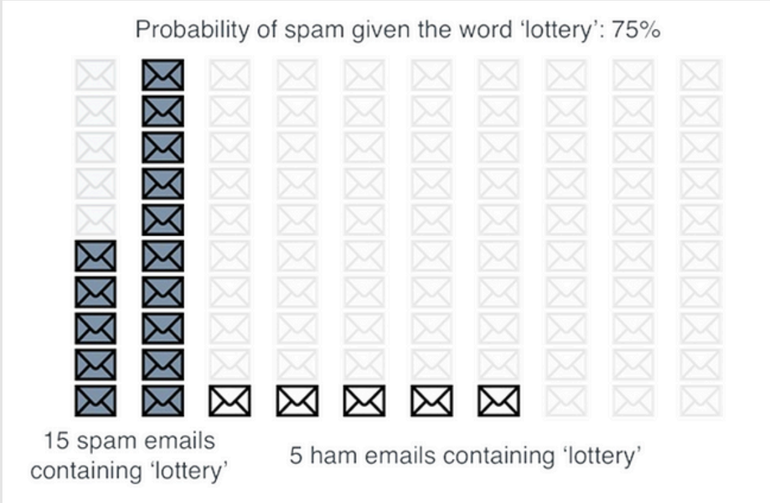
PRIOR: We look at our current inbox and count how many emails are spam and ham. Say there are 100 emails, of which 20 are spam and 80 are ham. Thus, 20% of the emails are spam. So, if we want to make a decent estimate, we can say that to the **best of our knowledge**, the probability that a new email is spam is 20%.



NAIVE BAYES ALGORITHM

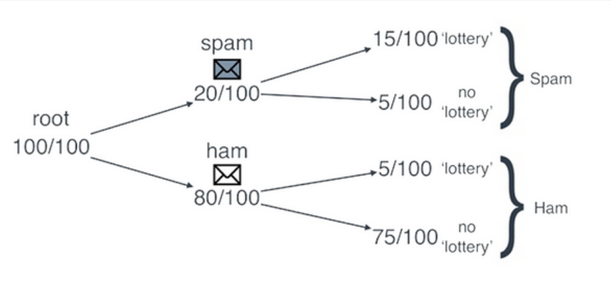
- **EVENT:** Let’s say we find that particular word, say, the word ‘lottery’, tends to appear in spam emails more than in ham emails. As a matter of fact, among the spam emails, it appears in 15 of them, while among the ham emails, it only appears in 5 of them.

- **POSTERIOR PROBABILITY:** Therefore, among the 20 emails containing the word ‘lottery’, 15 of them are spam, and 5 of them are ham. Thus, the probability that an email containing the word ‘lottery’ is spam is precisely 15/20, or 75%.

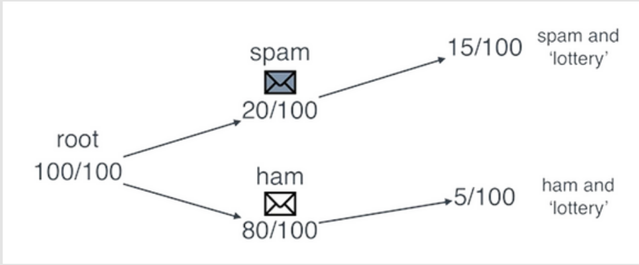


NAIVE BAYES ALGORITHM

- One way to visualize the example is with a tree of all the four possibilities, namely: that the email is spam or ham, and that it contains the word ‘lottery’ or not.



- We want to calculate the probability of an email being spam given that it contains the word ‘lottery’, we simply remove all the branches where the emails that don’t contain the word ‘lottery’



- What is the benefit of the diagram? Aside from making things simpler, the benefit is that normally, the information we have is based on probabilities and not on a number of emails.

NAIVE BAYES ALGORITHM

- **Some Notations**
- **RULE OF COMPLEMENTARY PROBABILITIES:** For an event, the complement of the event, denoted, is the event opposite to E. The probability of is 1 minus the probability of E. $P(E^c) = 1 - P(E)$
- P(spam)= The probability of an email being spam. P(ham)= The probability of an email being ham.
- The probability that a spam email contains the word ‘lottery’ is ¾. This can be read as the probability that an email contains the word ‘lottery’, **given that it is spam**, is ¾. This is a **conditional probability**. The condition is that the email is spam. We denote the condition by a vertical bar $P('lottery' \mid spam)$
- Now if we find the probabilities of two events happening at the same time, i.e *intersection* of events:
  - $P('lottery' \cap spam)$
  - $P(no\ 'lottery' \cap spam)$
  - $P('lottery' \cap ham)$
  - $P(no\ 'lottery' \cap ham)$
- Let’s look at some numbers. Out of 100 emails, 20 are spam, and ¾ of those contain the word "lottery"

Multiplying ⅕ (probability of spam) by ¾ (probability a spam email contains "lottery") gives 3/20 or 15%, which is the probability an email is both spam and contains "lottery." The probability that a spam email contains the word ‘lottery’ is precisely the conditional probability or the probability that an email contains the word ‘lottery’ given that it is a spam email. This gives rise to the multiplication rule for probabilities.

PRODUCT RULE OF PROBABILITIES: For events E and F, the probability of their intersection is precisely $P(E \cap F) = P(E \mid F) \cdot P(F)$

NAIVE BAYES ALGORITHM

We want to find This means, the probability that an email is spam given that it contains the word ‘lottery’. Among the four events we just studied, in only two of them does the word ‘lottery’ appear. Thus, we only need to consider those, namely:

$$P('lottery' \cap spam) = \frac{3}{20}$$
$$P('lottery' \cap ham) = \frac{1}{20}.$$

- The first should be the probability that an email is spam, and the second should be the probability that the email is ham. These two probabilities don’t add to 1.
- However, since the email contains the word "lottery," these are the only two possible scenarios, and their probabilities must sum to 1 while maintaining the same relative ratio. To fix this, we normalize by dividing each by their sum. In this case, the probabilities simplify to ¾ or 75% for spam and ¼ or 25% for ham.

$$P(spam \mid 'lottery') = \frac{P('lottery' \cap spam)}{P('lottery' \cap spam) + P('lottery' \cap ham)}.$$

$$P(spam \mid 'lottery') = \frac{P('lottery' \mid spam) \cdot P(spam)}{P('lottery' \mid spam) \cdot P(spam) + P('lottery' \mid ham) \cdot P(ham)}.$$

Formally defining Bayes theorem: For events E and F

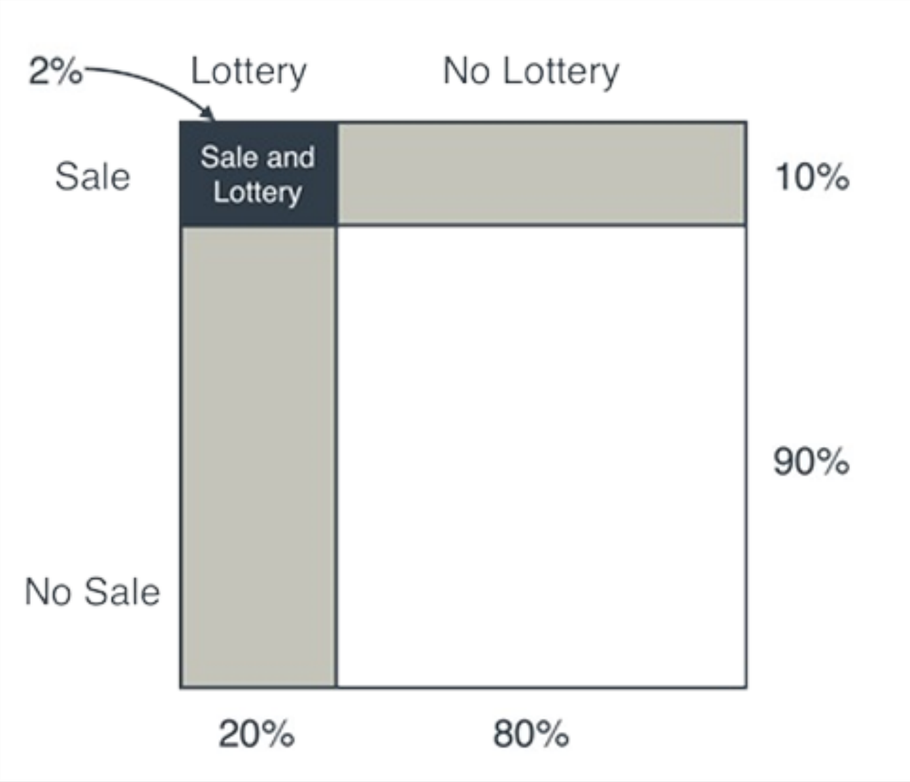
$$P(E \mid F) = \frac{P(F \mid E) \cdot P(E)}{P(F \mid E) \cdot P(E) + P(F \mid E^C) \cdot P(E^C)}.$$

NAIVE BAYES ALGORITHM

- Suppose we notice another word, ‘sale’, that also appears in many spam emails.
- And suppose we see that the probability of a spam mail having the word ‘sale’ is 60%
- How do we combine the two probabilities?
- An idea (a bad idea): let’s look at all the emails that have both words ‘lottery’ and ‘sale’. Count how many of them are spam and how many of them are ham, and find the probability based on those two numbers using Bayes theorem.
- Why is it bad? What if no emails in our dataset have the words ‘lottery’ and ‘sale’? We only have 100 emails, if only 20 of them have the word ‘lottery’ and 10 have the word sale, it could be that maybe only 1 or 2 have both words, or maybe none. The dataset is too small to draw accurate conclusions.
- What if we gather more data? That is always a good idea, but many times, we can’t, so we have to deal with the data we have.
- So, we make an assumption. A **naive assumption**: that the event containing the words ‘lottery’ and ‘sale’ are independent. This is, the appearance of one of the words in no way affects the appearance of the other one.
- The reason Naive Bayes is naive, is that it's simple: it treats all of the words independently, and this means it ignores word order and phrasing.

NAIVE BAYES ALGORITHM

Under this assumption, the percentage of emails that contain both words can be estimated as 2%, namely, the product of 20% and 10%.



PRODUCT RULE FOR INDEPENDENT PROBABILITIES: If two events and are independent, namely, the occurrence of one doesn't influence in any way the occurrence of the other one, then the probability of both of them happening is the product of the probabilities of each of the events. $P(E \cap F) = P(E) \cdot P(F)$

$$P(lottery \mid spam) = \frac{3}{4}$$
$$P(sale \mid spam) = \frac{3}{10}$$
$$P(lottery \cap sale \mid spam) = \frac{3}{4} \cdot \frac{3}{10} = \frac{9}{40} = 0.225$$

$$P(lottery \mid ham) = \frac{5}{80} = \frac{1}{16}$$
$$P(sale \mid ham) = \frac{4}{80} = \frac{1}{20}$$
$$P(lottery \cap sale \mid ham) = \frac{1}{16} \cdot \frac{1}{20} = \frac{1}{320} = 0.0625.$$

NAIVE BAYES ALGORITHM

So to summarize what we have done to calculate the probability an email containing the words ‘lottery’ and ‘sale’ is spam:

- $E = lottery \cap sale$
- $F = spam$,

$$P(spam \mid lottery \cap sale) = \frac{P(lottery \cap sale \mid spam) \cdot P(spam)}{P(lottery \cap sale \mid spam) \cdot P(spam) + P(lottery \cap sale \mid ham) \cdot P(ham)}.$$

$$P(lottery \cap sale \mid spam) = P(lottery \mid spam) \cdot P(sale \mid spam)$$

$$P(lottery \cap sale \mid ham) = P(lottery \mid ham) \cdot P(sale \mid ham)$$

$$P(spam \mid lottery \cap sale) = \frac{P(lottery \mid spam) \cdot P(sale \mid spam) \cdot P(spam)}{P(lottery \mid spam) \cdot P(sale \mid spam) \cdot P(spam) + P(lottery \mid ham) \cdot P(sale \mid ham) \cdot P(ham)}.$$

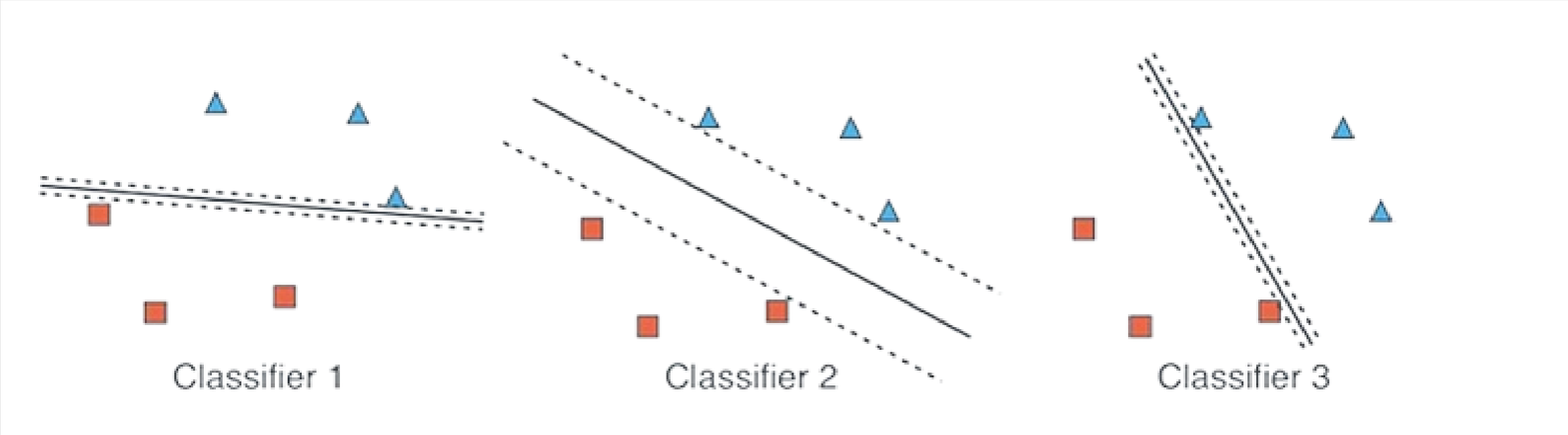
$$P(spam \mid lottery \cap sale) = \frac{\frac{3}{4} \cdot \frac{3}{10} \cdot \frac{1}{5}}{\frac{3}{4} \cdot \frac{3}{10} \cdot \frac{1}{5} + \frac{1}{16} \cdot \frac{1}{20} \cdot \frac{4}{5}} = 0.9863.$$

We began with Training Data and calculated Prior Probabilities, which are essentially our initial assumptions about whether a message is Normal or Spam, made without considering the message's content. Next, we combined these Prior Probabilities with the likelihood of each word appearing in either a Normal or Spam message. This allowed us to compute scores indicating the probability of the message being Spam or ham. Ultimately, we classify the message based on the highest score.

SUPPORT VECTOR MACHINES

- It is a linear classification model
- One difference between SVMs and other linear classifiers is that SVMs use two parallel lines instead of one.
- We draw our classifier as two parallel lines, as far apart from each other as possible.
- The maximum-margin approach is a key concept in Support Vector Machines (SVMs). It refers to how SVMs aim to find the decision boundary that maximizes the distance (margin) between the boundary and the closest data points from each class. These closest points are called support vectors.
- The margin is the space between the decision boundary (hyperplane) and the nearest data points of each class.
- The wider this margin, the more confident the classifier is in its prediction and the better it generalizes to unseen data.

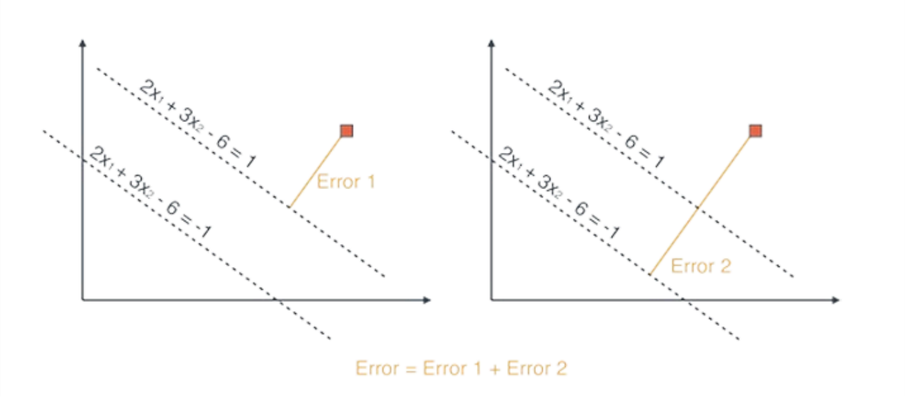
SUPPORT VECTOR MACHINES

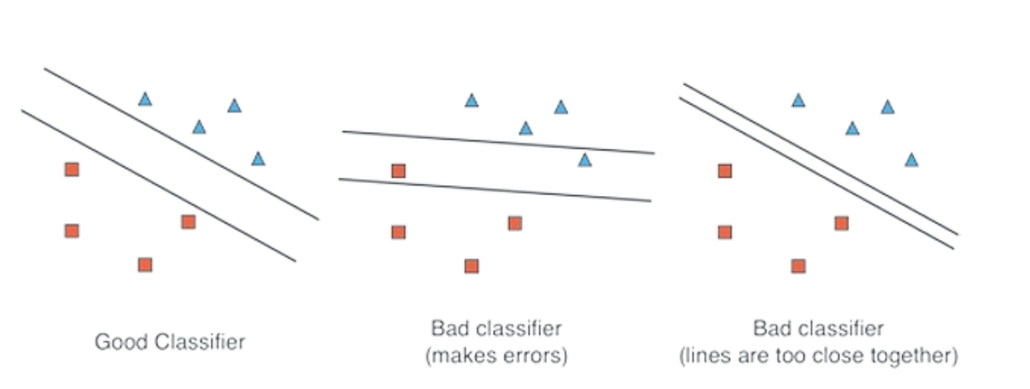


- Classifier 2 is the one where the parallel lines are farther from each other. This means that the main line in Classifier 2 is the one best located between the points.
- We need to build an error function that will create a classifier consisting of two lines
- Error = Classification Error + Distance Error

L+: $w_1x_1 + w_2x_2 + b = 1$, and
L-: $w_1x_1 + w_2x_2 + b = -1$.

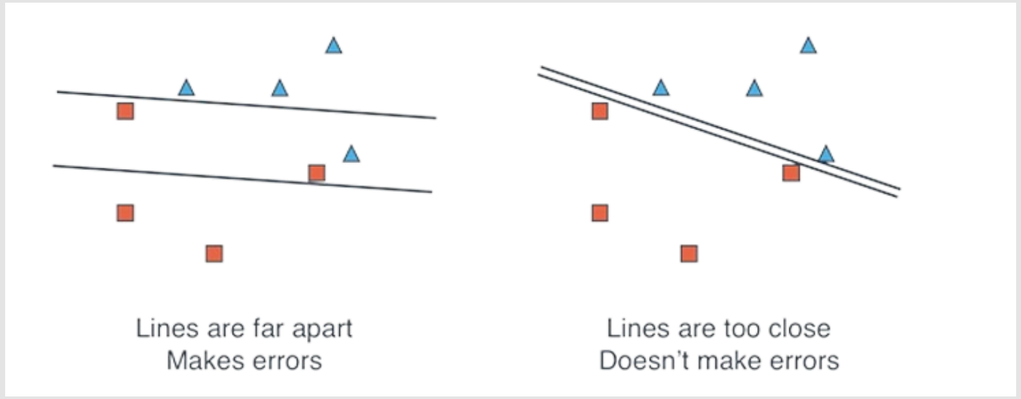
SUPPORT VECTOR MACHINES





- In an SVM, both lines have to classify the points well, so the classification error is a sum of the classification error of both models.
- If our lines have equations ax1 + bx2 = 1 and ax1 + bx2 = −1, then the error function to calculate distance error is a^2 + b^2
- Why? The perpendicular distance between the two lines is precisely 2/square_root(a^2 + b^2). When a^2 + b^2 is large, the distance is small and vice versa. The goal of SVM is to increase the distance. Hence, this error value punishes models with smaller distances.

SUPPORT VECTOR MACHINES

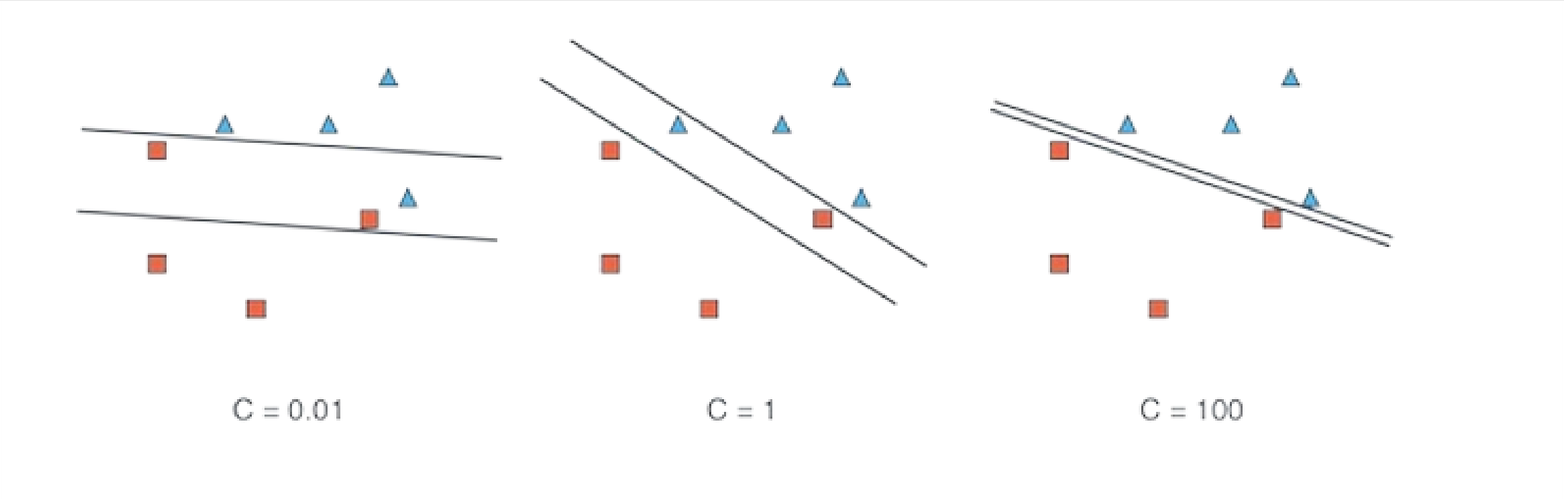


- Both of these classifiers have one pro and one con. The one in the left has the lines well spaced (pro), but it misclassifies some points (con). The one in the right has the lines too close together (con), but it classifies all the points correctly (pro).
- Which one do we prefer?
- The answer to this depends on the problem we are solving. Sometimes, we will want a classifier that makes as few errors as possible, even if the lines are too close, and sometimes, we will want a classifier that keeps the line apart, even if it makes a few errors.

SUPPORT VECTOR MACHINES

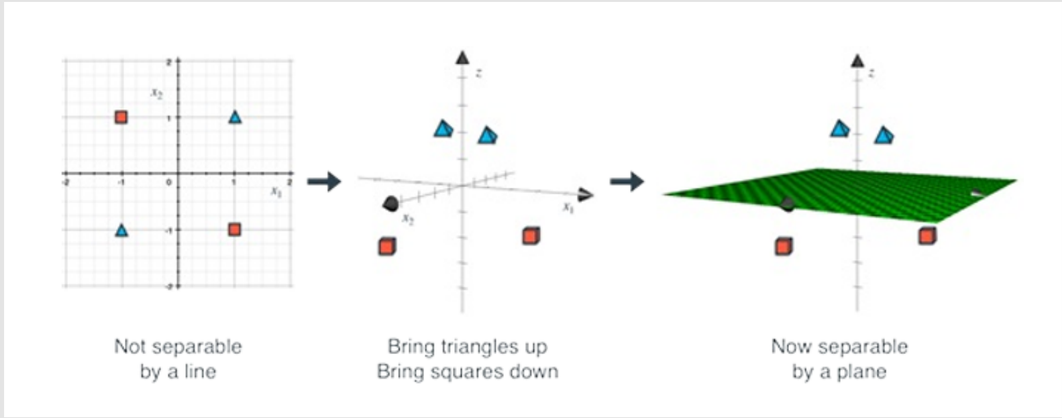
*Error formula = C * (Classification Error) + (Distance Error).*

- If C is a large value, then the error formula is dominated by the classification error, so our classifier will focus more on classifying the points correctly. If C is a small value, then the formula is dominated by the distance error, so our classifier will focus more on keeping the lines far apart.



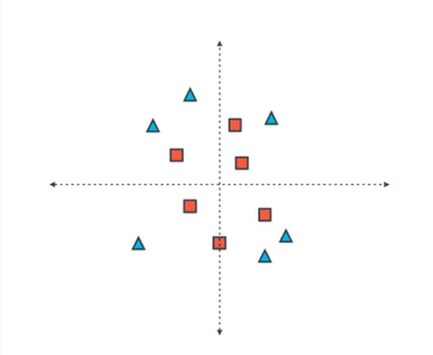
SUPPORT VECTOR MACHINES

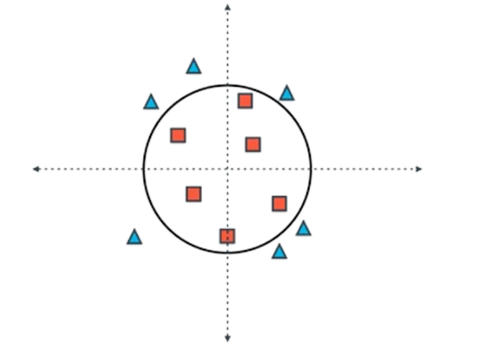
What if the points are not linearly separable?



- Imagine that your dataset is in two dimensions, which means your input has two columns. If you add a third column, you now have a three-dimensional dataset. If we cleverly do this, the points may be separable by a plane. This is the kernel method.
- The kernel is the name we give to a particular set of functions that will be used to add more columns to the dataset. Two important kernels are the polynomial kernel and the radial basis function (rbf) kernel.

SUPPORT VECTOR MACHINES





- The kernel method gives us a classifier with a circular boundary, which separates these points well.

DECISION TREES

- A classification model based on yes/no questions and is represented by a binary tree.
- The tree has a root, nodes, and leaves.
- **ROOT** The topmost node of the tree. It contains the first yes/no question.
- **NODE** Each yes/no question in our model is represented by a node, with two branches emanating from it (one for the ‘yes’ answer and one for the ‘no’ answer).
- **LEAF** When we reach a point where we don’t ask a question and instead make a decision, we have reached the leaf of the tree.
- Decision trees are widely used in machine learning for:

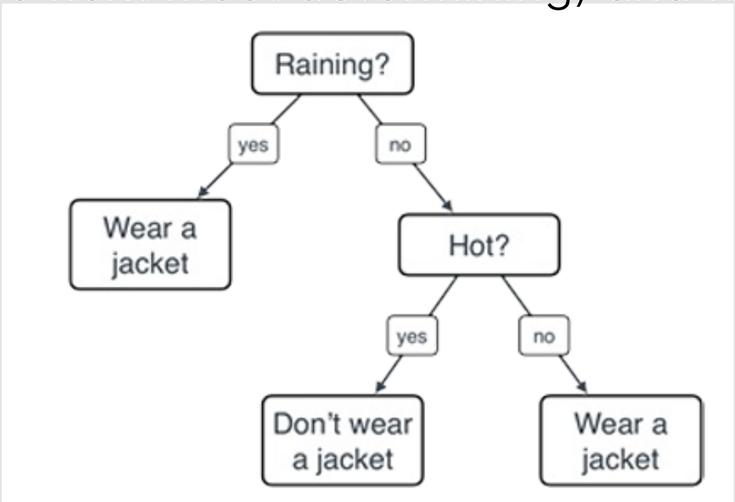
1. Recommendation systems (e.g., suggesting videos or products)
2. Spam classification (identifying spam emails)
3. Sentiment analysis (determining emotional tone)

DECISION TREES

- The algorithm that we use to build decision trees, in a nutshell, it works the following way:

1. Pick the most determining feature and divide the data according to that feature. In other words, among all the features in the data, pick the one that best divides the labels.

2. Either makes the prediction based on the answer to that question or picks another feature (the next most determining) and iterates.



DECISION TREES

Example: We need to recommend apps to users according to what they are likely to download

- Let’s say we are the App Store, or Google Play, and we want to recommend to our users the app that they are most likely to download. We have three apps in our store: Atom Count: An app that counts the number of atoms in your body. · Beehive Finder: An app that maps your location and finds the closest beehives. · Check Mate Mate: An app for finding chess players in your area.

DECISION TREES

Our Dataset:

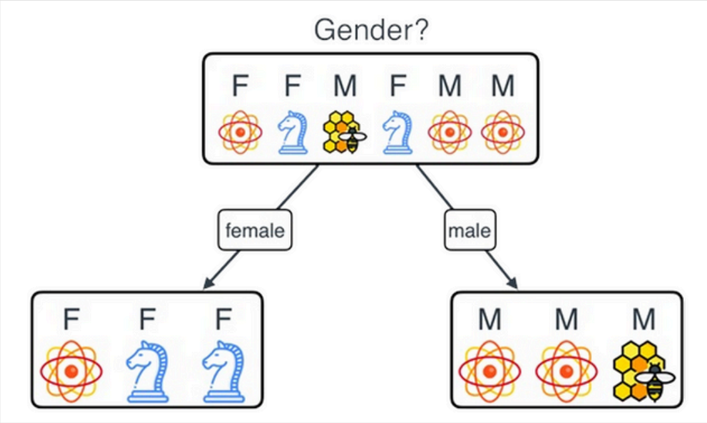
Gender	Age	App
F	15	Atom Count
F	25	Check Mate Mate
M	32	Beehive Finder
F	35	Check Mate Mate
M	12	Atom Count
M	14	Atom Count

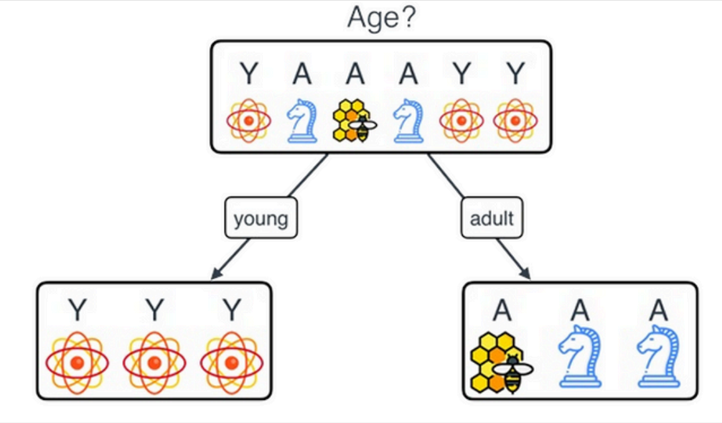
The building blocks of the model are questions so lets modify the dataset a bit:

Gender	Age	App
F	young	Atom Count
F	adult	Check Mate Mate
M	adult	Beehive Finder
F	adult	Check Mate Mate
M	young	Atom Count
M	young	Atom Count

DECISION TREES

- Based on the data we can ask 2 questions. “Is the user female?” or “Is the user young?”
- We need one of these to use as the root of the tree. Which one should we pick? We should pick the one that determines the app they downloaded best. To decide which question is better at this, let’s compare them.





- It appears that age is better since it has picked up on the fact that all three young people downloaded Atom Count.

- But we need the computer to figure out that age is a better feature, so we need to give it some numbers to compare.
- We’ll look at three ways to attach a number to each of the features, in order to compare them. These are accuracy, Gini impurity, and Gini gain.

DECISION TREES

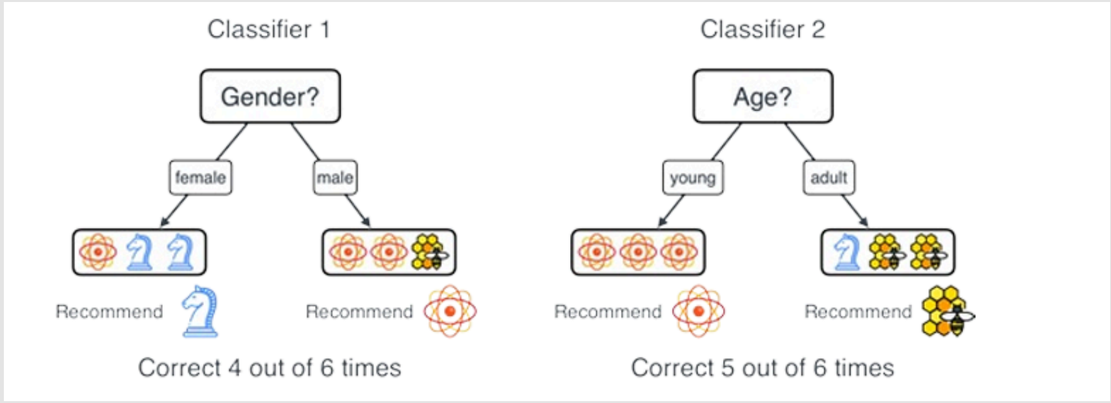
- Let's start with **Accuracy**.
- To decide the root, let's assume we are allowed to ask only one question.

Classifier 1: Uses the gender. In this case, for ‘F’, there are 2 Check Mate Mates and one Beehive Finder. We’ll take the more popular one, which is Check Mate Mate. Similarly, for ‘M’. Now, if a user is ‘F’, We recommend Check Mate Mate; if ‘M’, we recommend Atom Count. Now, we calculate the accuracy of our prediction.

Classifier 2: Similarly calculate the accuracy

- We take the feature with higher accuracy as the root

We have decided that if a person is young, we’ll recommend them Atom Count. This means at that point we stop. As we saw before, this is called a leaf. If they are adults, however, we can recommend them Check Mate Mate, but we can do better.

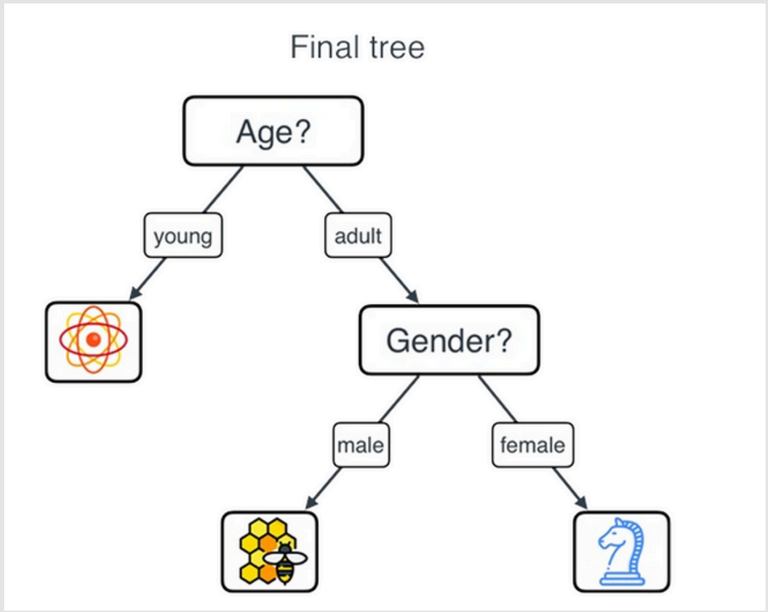


DECISION TREES

Original table, reduced to only the adults

F	adult	Check Mate Mate
M	adult	Beehive Finder
F	adult	Check Mate Mate

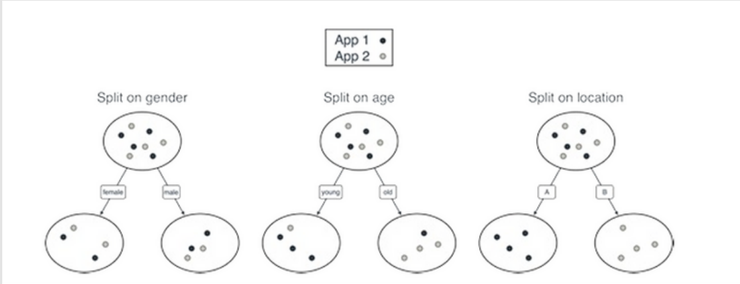
Notice that if we divide the users by gender, we get that ‘F’ downloaded Check Mate Mate, while the “M’ downloaded Beehive Finder. Therefore, we can just add this branch to our tree

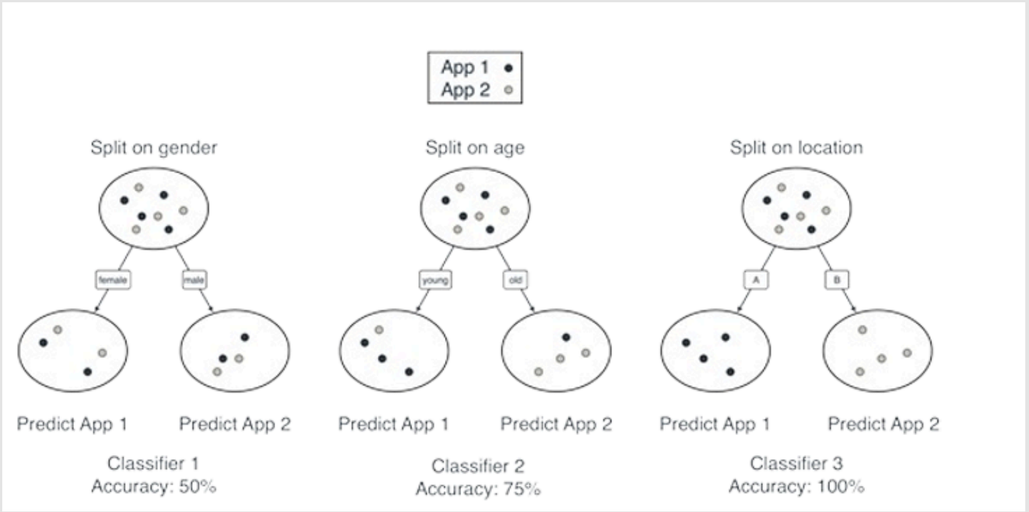


DECISION TREES

Now, let's take a dataset with three features: gender, age, and location. The location is where each user lives, and it can only be one of two cities: city A and city B. Furthermore, there are only two apps to download instead of three, and their names are App 1 and App 2.

Gender	Age	Location	App
Female	Young	A	1
Male	Young	A	1
Female	Young	A	1
Male	Adult	A	1
Female	Young	B	2
Male	Adult	B	2
Female	Adult	B	2
Male	Adult	B	2





DECISION TREES

Gini impurity:

- The idea behind the concept of impurity is to come up with a number that is low if a set is very homogeneous, and high if a set is very heterogeneous.

Set 1: App 1, App 1, App 1, App 1.
Set 2: App 1, App 1, App 1, App 2.
Set 3: App 1, App 1, App 2, App 2.
Set 4: App 1, App 2, App 2, App 2.
Set 5: App 2, App 2, App 2, App 2.

Sets 1 and 5 are completely homogeneous since all users downloaded the same app. These two would have a very low impurity index. Set 3 is the most diverse since it has two users downloading App 1 and two users downloading App 2. This set would have a high impurity index. Sets 1 and 4 are in between, so they would have a medium impurity index.

DECISION TREES

For convenience, we'll think of App 1 as a black ball, and App 2, as a white ball.

Our sets are then the following:

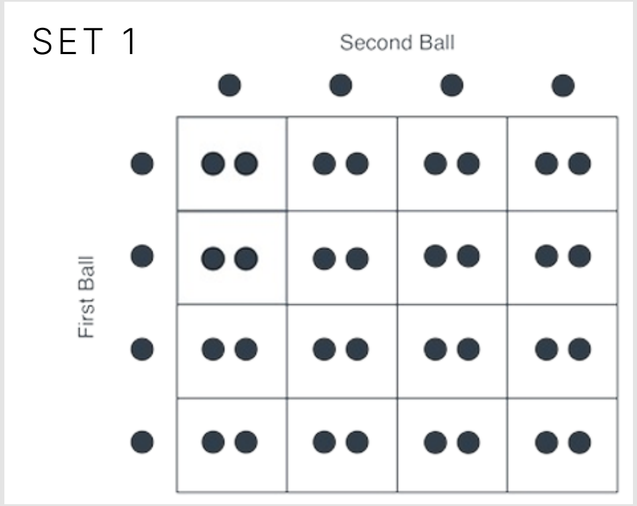
Set 1: {black, black, black, black}

Set 2: {black, black, black, white}

Set 3: {black, black, white, white}

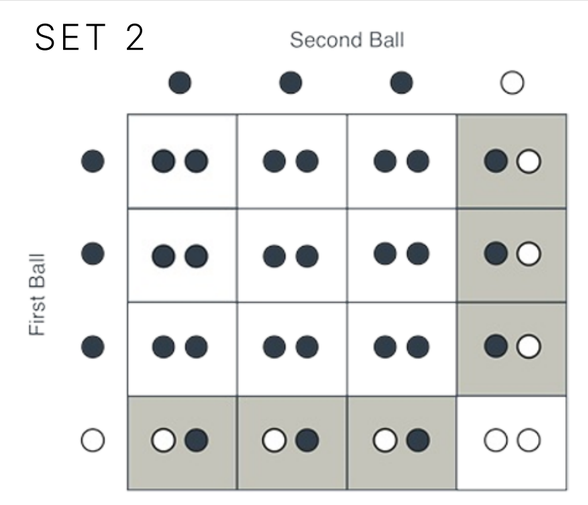
Set 4: {black, white, white, white}

Set 5: {white, white, white, white}

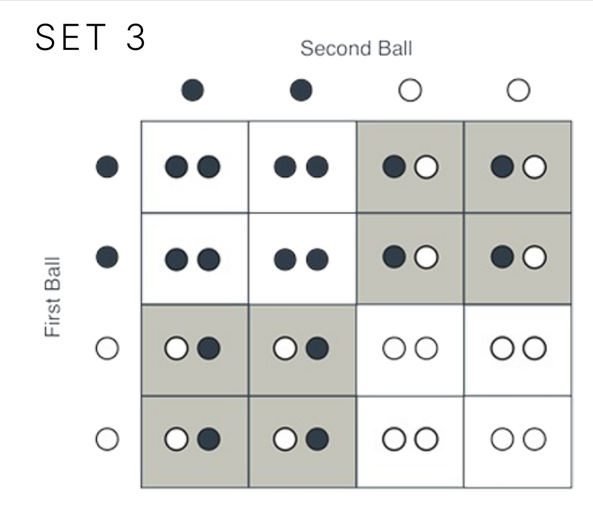


If we pick our balls from a bag containing four black balls, we get 16 scenarios. All of them give us the pair (black, black). Thus, the probability of obtaining two balls of different colors is zero. Therefore, we define the Gini impurity of Set 1 to be 0.

DECISION TREES



The probability of picking two balls of different colors is 6/16. We then define the Gini impurity of Set 2 to be 6/16 (or 0.375)



The probability of picking two balls of different colors is 8/16. We then define the Gini impurity of Set 2 to be 8/16 (or 0.5).

DECISION TREES

Let’s say we have N balls that could be of k colours, 1, 2,..., k. Say there are ai balls of colour i (therefore, a1 + a2 + ... + ak = N). The probability of picking a ball of colour i is ai/N. Therefore, the probability of picking two balls of colour i is pi^2.

Gini Impurity Index = P(picking two balls of different color)

= 1 - P(picking two balls of the same color)

= $1 - p_1^2 - p_2^2 - \ldots - p_N^2$

P(Both balls are color 1)

P(Both balls are color 2)

P(Both balls are color N)

For example:

Gini Impurity Index = P(picking two balls of different color)

= $1 - (\frac{3}{6})^2 - (\frac{2}{6})^2 - (\frac{1}{6})^2$

P(Both balls are black)

P(Both balls are grey)

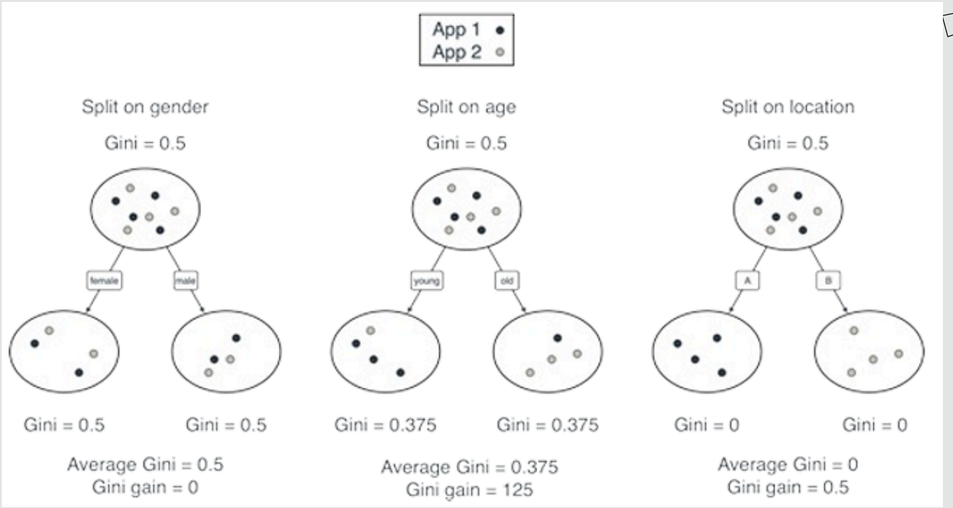
P(Both balls are white)

DECISION TREES

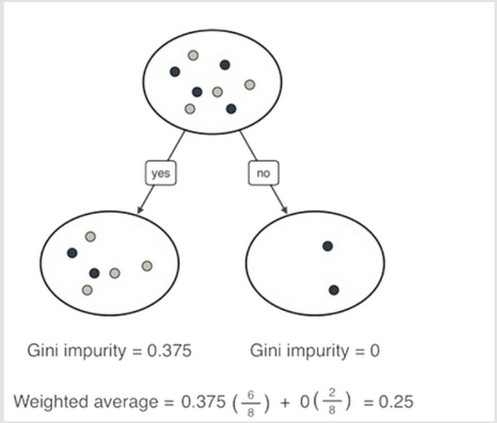
Gini Gain:
Procedure to measure a split:

- Calculate the Gini impurity index of the root.
- Calculate the average of the Gini impurity indices of the two leaves.
- Subtract them, to obtain the gain on Gini impurity index

In the above case we used the average Gini impurity index. But what is more useful is the weighted average Gini impurity index. Because we not only want our split as pure as possible we also would like to make sure that one feature doesn't overly influence the final result.



Example:



DECISION TREES

Let’s abbreviate our apps as follows:
A: Atom Count
B: Beehive Finder
C: Check Mate Mate

Our goal is to find the Gini impurity index of the set {A, C, B, C, A, A}(Root). Probabilities:
·Letter A: ½ (since there are 3 A’s out of 6 letters).
·Letter B: ⅙ (since there is 1 B out of 6 letters).
·Letter C: ⅓ (since there are 2 C’s out of 6 letters).

Gini impurity index of (A, C, B, C, A, A} = 1 - (1/2)^2 - (1/6)^2 - (1/3)^2 = 0.611

Splitting by gender gives us the two following sets: · Females: {A, C, C} · Males: {B, A, A}

The Gini impurity indices of these two are the following:
· Females: 1 - (1/3)^2 - (2/3)^2 = 0.444
· Males: 1 - (1/3)^2 - (2/3)^2 = 0.444
Therefore, the average Gini impurity index of this splitting is 0.444. Since the Gini impurity of the root was 0.611, we conclude that the Gini gain is: Gini gain = 0.611 - 0.444 = 0.167

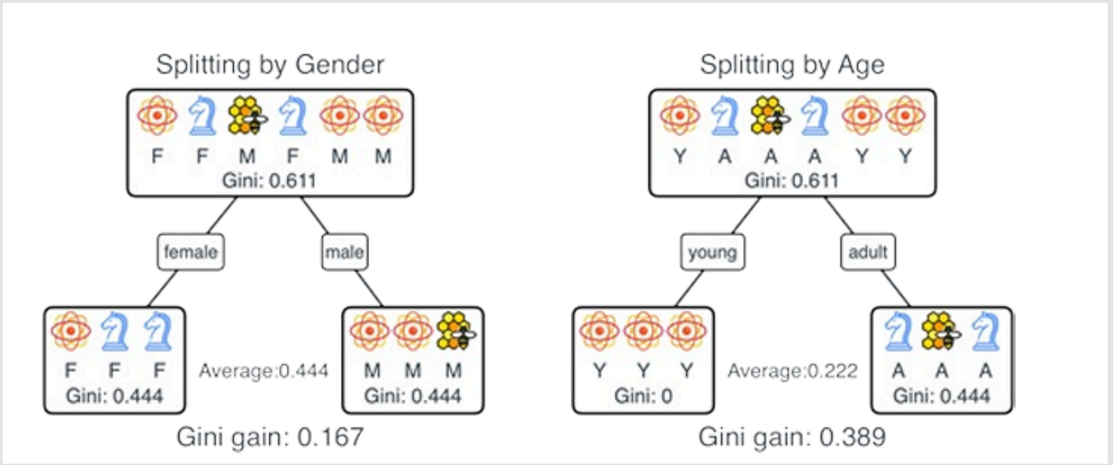
DECISION TREES

As we've seen before, splitting by age gives us the two following sets:

· Young: {A, A, A}
· Adults: {C, B, C}

The Gini impurity indices of these two are the following:

· Young: 1 - (3/3)^2 = 0
· Adults: 1 - (1/3)^2 - (2/3)^2 = 0.444



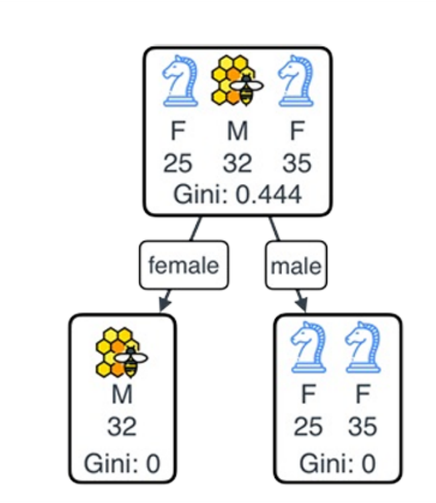
Therefore, the average Gini impurity index of this splitting is 0.222 (the average of 0 and 0.444).
Since the Gini impurity of the root was 0.611,
we conclude that the Gini gain is: Gini gain = 0.611-0.222 = 0.389

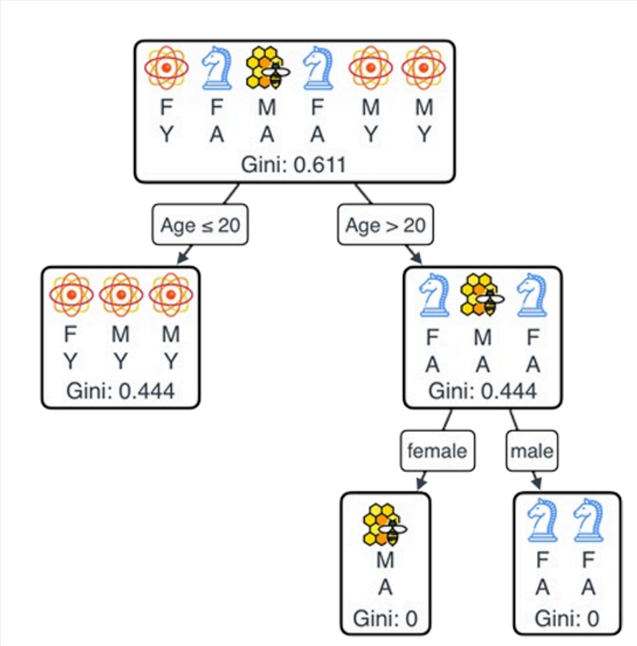
We decided to split by age, resulting in two nodes:

- *Left node*: Everyone downloaded Atom Count, with a Gini impurity of 0. This node can't be split further and becomes a leaf, as we can recommend Atom Count to all users here.
- *Right node:* One person downloaded Beehive Finder, and two downloaded Check Mate Mate, with a Gini impurity of 0.444. This node can still be split by gender since all users are adults.

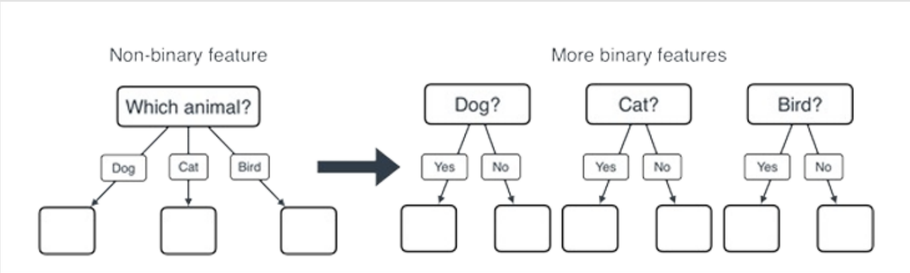
DECISION TREES

Performing the split:





What if we have more than 2 features?



Non-binary feature?

Animal
Dog
Cat
Bird
Dog
Bird

One-hot encoding

Dog?	Cat?	Bird?
1	0	0
0	1	0
0	0	1
1	0	0
0	0	1

DECISION TREES

Why did we choose 20 as the age threshold?

Splitting	Labels	Accuracy	Gini gain
{12} {14, 15, 25, 32, 40}	A A, A, C, B, C	3/6	0.078
{12, 14} {15, 25, 32, 40}	A, A A, C, B, C	4/6	0.194
{12, 14, 15} **{25, 32, 40}**	**A, A, A** **C, B, C**	**5/6**	**0.389**
{12, 14, 15, 25} {32, 40}	A, A, A, C B, C	4/6	0.194
{12, 14, 15, 25, 32} {40}	A, A, A, C, B C	4/6	0.144

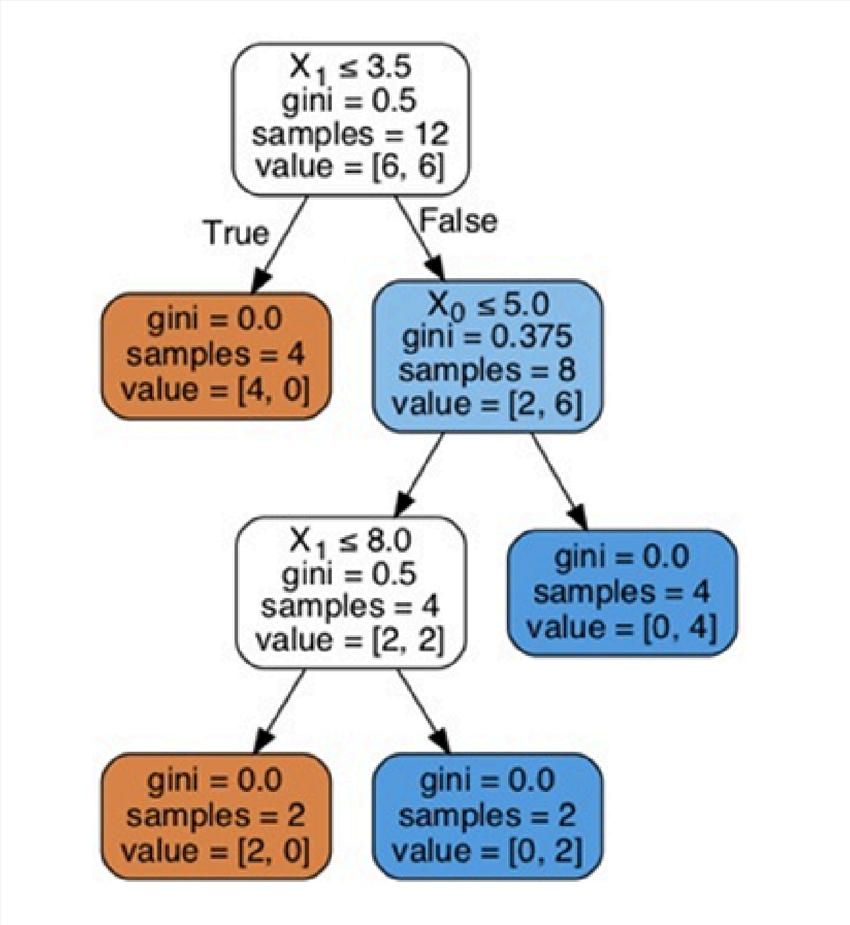
DECISION TREES

Exercise: Find the decision tree for the spam dataset

Lottery	Sale	Spam
7	1	No
3	2	No
3	9	No
1	3	No
2	6	No
4	7	No
1	9	Yes
3	10	Yes
6	5	Yes
7	8	Yes
8	4	Yes
9	6	Yes

DECISION TREES

Exercise: Find the decision tree for the spam dataset



x0 : The first column in our features

X1 : The second column in our features

Gini: The Gini impurity index of the elements that fall in each node.

Samples: The number of samples on each node.

Value: This vector shows the distribution of labels in each node.

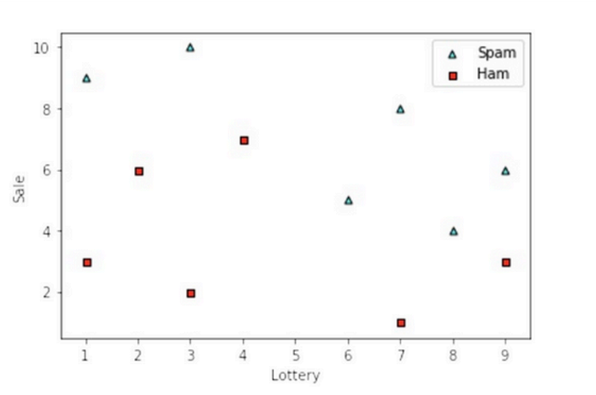
RANDOM FOREST

- Random forests are an ensemble learning method that combines multiple decision trees to improve classification or regression tasks.
- The core idea behind random forests is that while individual decision trees may be prone to overfitting, combining many trees (each slightly different from one another) leads to a model that is more robust and generalizes better to unseen data.
- When we have several classifiers that classify our data well, but not great, and we'd like to combine them into a super-classifier that classifies our data very well. This discipline is called ensemble learning.
- **TERMS:**

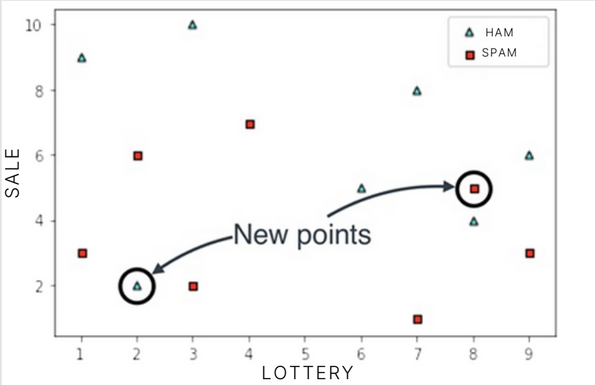
Bagging (Bootstrap AGGregatING) : We train a bunch of different classifiers on different random subsets of our data. We join them into a big classifier, which makes predictions by voting. In other words, the resulting classifier predicts what the majority of the other classifiers predict.

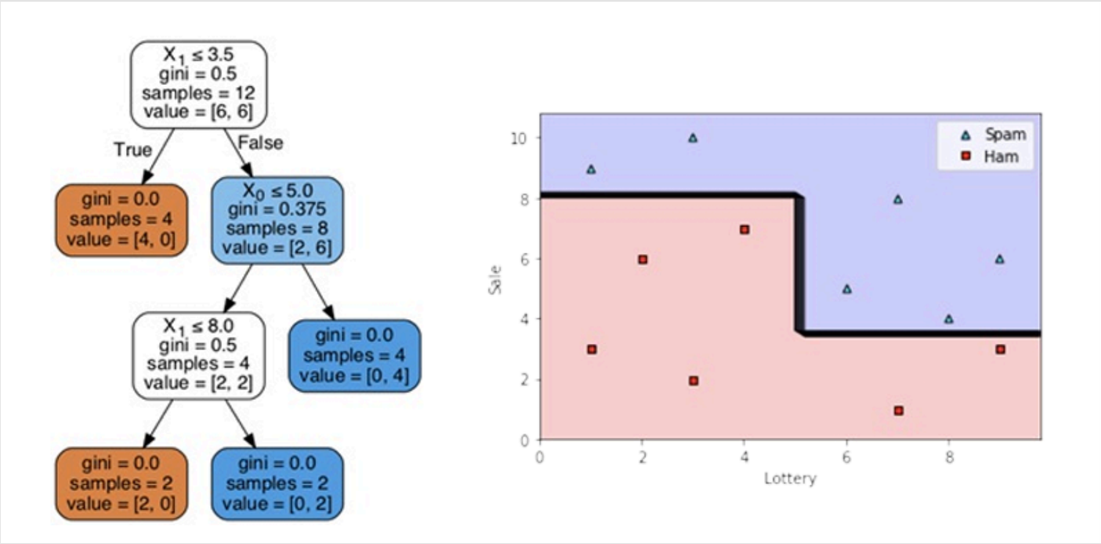
A canonical example of a bagging model is random forests. Simply put, you pick a random subset of your data and build a decision tree that fits that subset. Then you do the same thing with another random subset of data. You continue in this fashion, building a set of decision trees. Then you build a classifier joining all these decision trees by making them vote. Since the trees were built on random subsets of you data, we call this a random forest.

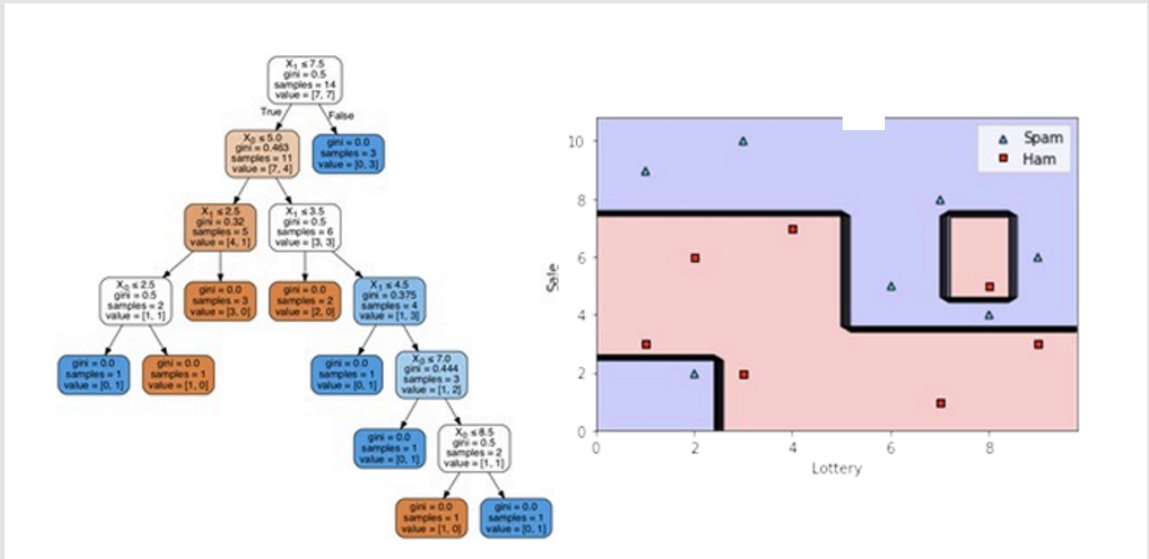
RANDOM FOREST



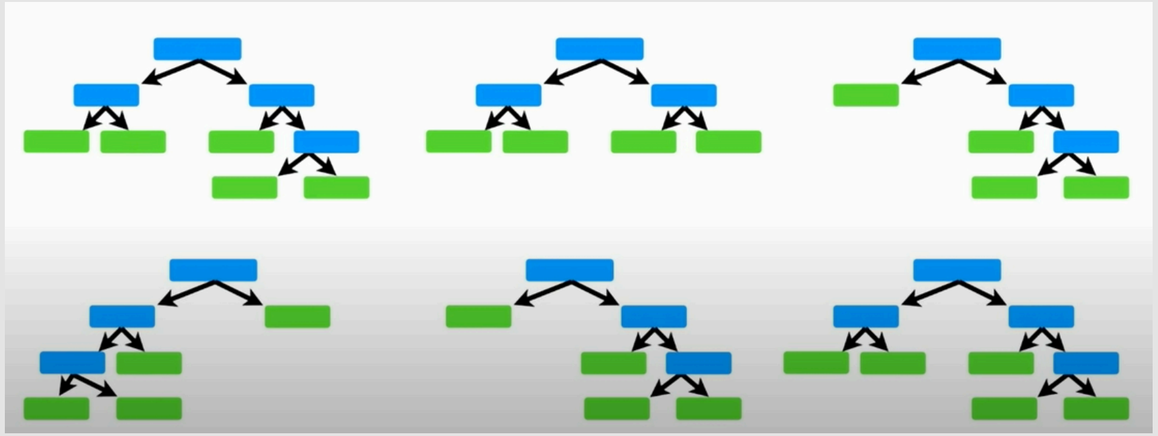
- Why do we need random forests?
- Decision trees are very prone to overfitting







RANDOM FOREST



Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

How Random Forests Work:

1. **Bootstrapping (Random Sampling):**

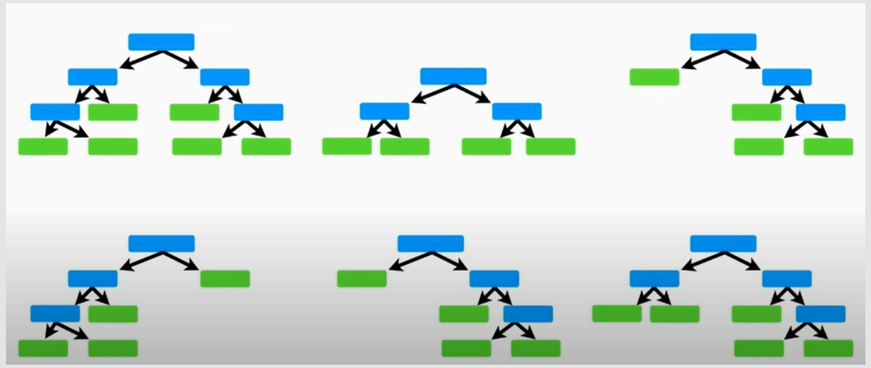
- The first step in creating a random forest is to generate multiple decision trees. Each tree is trained on a different subset of the training data. These subsets are created using a method called bootstrapping, where samples are drawn with replacements from the original dataset. This means some data points may appear multiple times in one subset while others may be left out.
- This process ensures that each tree is trained on a slightly different dataset, which makes each tree unique.

RANDOM FOREST

2. Random Feature Selection:

- When building each decision tree, instead of considering all the features (variables) for splitting at each node, a random subset of features is selected. This forces the trees to be different from each other, as each tree will make different splitting decisions based on the features they are allowed to consider.
- This step reduces the correlation between the trees and ensures that the random forest can capture different aspects of the data.

3. Go back to Step 1 and repeat multiple times. Make a new bootstrapped dataset and a build a tree considering a subset of variables each time.



RANDOM FOREST

4. Aggregating the Results (Voting or Averaging):
   - For classification tasks, each tree in the forest gives a "vote" for the class it predicts. The random forest chooses the class that receives the majority of votes across all the trees.
   - For regression tasks, the final prediction is the average of all the individual tree predictions.

-> This bootstrapping of data and then aggregating the results is bagging
-> Since each time only a subset of the data is sampled, there will be some samples that are never chosen, roughly 1/3rd. These are called out-of-bag samples. We use them for testing the accuracy of our model. We'll run each out-of-bag through the random forest and measure the accuracy of the model by the proportion of out-of-bag samples correctly classified.
-> The proportion of out-of-bag samples incorrectly classified is the out-of-bag error
-> We can use this error to determine the optimal number of features to randomly pick during the construction of the individual trees.

THANK YOU