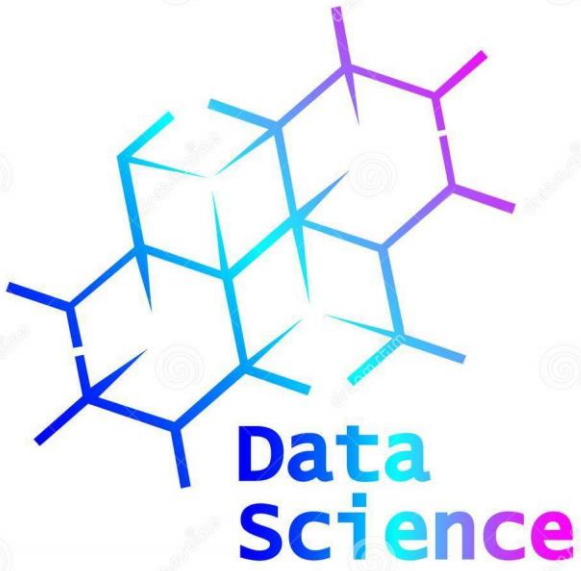


BCA-6TH SEM

Benefits and uses of data science and big data

Data science and big data are used almost everywhere in both commercial and non-commercial settings. The number of use cases is vast, and the examples we'll provide throughout this book only scratch the surface of the possibilities. Commercial companies in almost every industry use data science and big data to gain insights into their customers, processes, staff, completion, and products. Many companies use data science to offer customers a better user experience, as well as to cross-sell, up-sell, and personalize their offerings. A good example of this is Google AdSense, which collects data from internet users so relevant commercial messages can be matched to the person browsing the internet. MaxPoint (http://maxpoint.com/us) Benefits and uses of data science and big data 3 is another example of real-time personalized advertising. Human resource professionals use people analytics and text mining to screen candidates, monitor the mood of employees, and study informal networks among coworkers. People analytics is the central theme in the book Moneyball: The Art of Winning an Unfair Game. In the book (and movie) we saw that the traditional scouting process for American baseball was random, and replacing it with correlated signals changed everything. Relying on statistics allowed them to hire the right players and pit them against the opponents where they would have the biggest advantage. Financial institutions use data science to predict stock markets, determine the risk of lending money, and learn how to attract new clients for their services. At the time of writing this book, at least 50% of trades worldwide are performed automatically by machines based on algorithms developed by quants, as data scientists who work on trading algorithms are often called, with the help of big data and data science techniques. Governmental organizations are also aware of data's value. Many governmental organizations not only rely on internal data scientists to discover valuable information, but also share their data with the public. You can use this data to gain insights or build data-driven applications. Data.gov is but one example; it's the home of the US Government's open data. A data scientist in a governmental organization gets to work on diverse projects such as detecting fraud and other criminal activity or optimizing project funding. A well-known example was provided by Edward Snowden, who leaked internal documents of the American National Security Agency and the British Government Communications Headquarters that show clearly how they used data science and big data to monitor millions of individuals. Those organizations collected 5 billion data records from widespread applications such as Google Maps, Angry Birds, email, and text messages, among many other data sources. Then they applied data science techniques to distill information. Nongovernmental organizations (NGOs) are also no strangers to using data. They use it to raise money and defend their causes. The World Wildlife Fund (WWF), for instance, employs data scientists to increase the effectiveness of their fundraising efforts. Many data scientists devote part of their time to helping NGOs, because NGOs often lack the resources to collect data and employ data scientists. DataKind is one such data scientist group that devotes its time to the benefit of mankind. Universities use data science in their research but also to enhance the study experience of their students. The rise of massive open online courses (MOOC) produces a lot of data, which allows universities to study how this type of learning can complement traditional classes. MOOCs are an invaluable asset if you want to become a data scientist and big data professional, so definitely look at a few of the better-known ones: Coursera, Udacity, and edX. The big data and data science landscape changes quickly, and MOOCs allow you to

stay up to date by following courses from top universities. If you aren't acquainted with them yet, take time to do so now; you'll come to love them as we have

Facets of data

In data science and big data you'll come across many different types of data, and each of them tends to require different tools and techniques. The main categories of data are these:

■ Structured

■ Unstructured

■ Natural language

■ Machine-generated

■ Graph-based

■ Audio, video, and images

■ Streaming

Let's explore all these interesting data types

The data science process

The data science process typically consists of six steps, as you can see in the mind map in figure 1.5. We will introduce them briefly here and handle them

Data science process

1: Setting the research goal

2: Retrieving data

3: Data preparation

4: Data exploration

5: Data modeling

6: Presentation and automation

We'll define statistical inference as the process of generating conclusions about a population from a noisy sample. Without statistical inference we're simply living within our data. With statistical inference, we're trying to generate new knowledge. Knowledge and parsimony, (using simplest reasonable models to explain complex phenomena), go hand in hand. Probability models will serve as our parsimonious description of the world. The use of probability models as the connection between our data and a populations represents the most effective way to obtain inference.

Motivating example: who's going to win the election? In every major election, pollsters would like to know, ahead of the actual election, who's going to win. Here, the target of estimation (the estimand) is clear, the percentage of people in a particular group (city, state, county, country or other electoral grouping) who will vote for each candidate. We can not poll everyone. Even if we could, some polled may change their vote by the time the election occurs.

How do we collect a reasonable subset of data and quantify the uncertainty in the process to produce a good guess at who will win?

Motivating example, predicting the weather When a weatherman tells you the probability that it will rain tomorrow is 70%, they're trying to use historical data to predict tomorrow's weather - and to actually attach a probability to it. That probability refers to population.

Motivating example, brain activation An example that's very close to the research I do is trying to predict what areas of the brain activate when a person is put in the fMRI scanner. In that case, people are doing a task while in the scanner. For example, they might be tapping their finger. We'd like to compare when they are tapping their finger to when they are not tapping their finger and try to figure out what areas of the brain are associated with the finger tapping.

Summary notes These examples illustrate many of the difficulties of trying to use data to create general conclusions about a population. Paramount among our concerns are:

• Is the sample representative of the population that we'd like to draw inferences about? • Are there known and observed, known and unobserved or unknown and unobserved variables that contaminate our conclusions? • Is there systematic bias created by missing data or the design or conduct of the study? • What randomness exists in the data and how do we use or adjust for it? Here randomness can either be explicit via randomization or random sampling, or implicit as the aggregation of many complex unknown processes. • Are we trying to estimate an underlying mechanistic model of phenomena under study? Statistical inference requires navigating the set of assumptions and tools and subsequently thinking about how to draw conclusions from data. Introduction 4 The goals of inference You should recognize the goals of inference. Here we list five examples of inferential goals.

1. Estimate and quantify the uncertainty of an estimate of a population quantity (the proportion of people who will vote for a candidate). 2. Determine whether a population quantity is a benchmark value ("is the treatment effective?"). 3. Infer a mechanistic relationship when quantities are measured with noise ("What is the slope for Hooke's law?") 4. Determine the impact of a policy? ("If we reduce pollution levels, will asthma rates decline?") 5. Talk about the probability that something occurs.

The tools of the trade Several tools are key to the use of statistical inference. We'll only be able to cover a few in this class, but you should recognize them anyway.

1. Randomization: concerned with balancing unobserved variables that may confound inferences of interest. 2. Random sampling: concerned with obtaining data that is representative of the population of interest. 3. Sampling models: concerned with creating a model for the sampling process, the most common is so called "iid". 4. Hypothesis testing: concerned with decision making in the presence of uncertainty. 5. Confidence intervals: concerned with quantifying uncertainty in estimation. 6. Probability models: a formal connection between the data and a population of interest. Often probability models are assumed or are approximated. 7. Study design: the process of designing an experiment to minimize biases and variability. 8. Nonparametric bootstrapping: the process of using the data to, with minimal probability model assumptions, create inferences. 9. Permutation, randomization and exchangeability testing: the process of using data permutations to perform inferences. Different thinking about probability leads to different styles of inference We won't spend too much time talking about this, but there are several different styles of inference. Two broad categories that get discussed a lot are: Introduction 5 1. Frequency probability: is the long run proportion of times an event occurs in independent, identically distributed repetitions. 2. Frequency style inference: uses frequency interpretations of probabilities to control error rates. Answers questions like "What should I decide given my data controlling the long run proportion of mistakes I make at a tolerable level." 3. Bayesian probability: is the probability calculus of beliefs, given that beliefs follow certain rules. 4. Bayesian style inference: the use of Bayesian probability representation of beliefs to perform inference. Answers questions like "Given my subjective beliefs and the objective information from the data, what should I believe now?" Data scientists tend to fall within shades of gray of these and various other schools of inference. Furthermore, there are so many shades of gray between the styles of inferences that it is hard to pin down most modern statisticians as either Bayesian or frequentist. In this class, we will primarily focus on basic sampling models, basic probability models and frequency style analyses to create standard inferences. This is the most popular style of inference by far. Being data scientists, we will also consider some inferential strategies that rely heavily on the observed data, such as permutation testing and bootstrapping. As probability modeling will be our starting point, we first build up basic probability as our first task. Exercises 1. The goal of statistical inference is to? • Infer facts about a population from a sample. • Infer facts about the sample from a population. • Calculate sample quantities to understand your data. • To torture Data Science students. 2. The goal of randomization of a treatment in a randomized trial is to? • It doesn't really do anything. • To obtain a representative sample of subjects from the population of interest. • Balance unobserved covariates that may contaminate the comparison between the treated and control groups. • To add variation to our conclusions. 3. Probability is a? • Population quantity that we can potentially estimate from data. • A data quantity that does not require the idea of a population.

Population:

A complete collection of the objects or measurements is called the population or else everything in the group we want to learn about will be termed as population. or else In statistics population is the entire set of items from which data is drawn in the statistical study. It can be a group of individuals or a set of items.

Population is the entire group you want to draw conclusions about.
The population is usually denoted with **N**

1. The number of citizens living in the State of Rajasthan represents a population of the state
2. All the chess players who have FIDE rating represents the population of the chess fraternity of the world
3. the number of planets in the entire universe represents the planet population of the entire universe
4. The types of candies and chocolates are made in India.

population mean is usually denoted by the Greek letter μ

μ **(population mean) =** $\sum_{i=1}^{N}(xi)$ / N(total population)

For example: let us assume that there are 5 employees in my company, so 5 people is a complete set hence it will represent the population of my company

If I wanna find the average age of my company then I will simply add their ages and divide it by N which is the number of population

ages = {23,45,12,34,22}

$$\mu = \sum_{i=1}^{5}(xi)/5$$
$$= (23 + 45 + 12 + 34 + 22) / 5$$

the results say that the average age of my company is 27.2 years

so this is what we call the population mean. Here the population was quite less so the calculation of the population means was an easy task ut what if we want to calculate the average height of the Indians, it would be next to impossible task because every second a person is taking birth and a person is dying. So even if it is not impossible it would be a difficult task.

Since in this case and many others it is impossible to observe the entire statistical population due to the time constraints, constraints on geographical accessibility and the constraints on the researcher resources a researcher would instead observe a sample of the same population in order to attempt to learn something about the population as a whole.
This brings us to the topic of what is sample??

Sample:

A sample represents a group of the interest of the population which we will use to represent the data. The sample is an unbiased subset of the population in which we represent the whole data. A sample is a group of the elements actually participating in the survey or study.

A sample is the representation of the manageable size. samples are collected and stats are calculated from the sample so one can make interferences or extrapolations from the sample. This process of collecting info from the sample is called **sampling.**

The sample is denoted by the ***n***

1. 500 people from a total population of the Rajasthan state will be considered as a sample
2. 143 total chess players from all total number of chess players will be considered as a sample

Sample mean is denoted by x –

$$x\ (\textbf{\textit{sample mean}}) = \sum_{i=1}^{n} (xi)/n \text{ (total sample)}$$

for example:

1. The total number of users of geeksforgeeks is the population and, all student accounts of the website is a sample.
2. All the FIDE rated chess players in the population and players having a rating of more than 1700 is a sample.

- ***How to collect data from the population?***

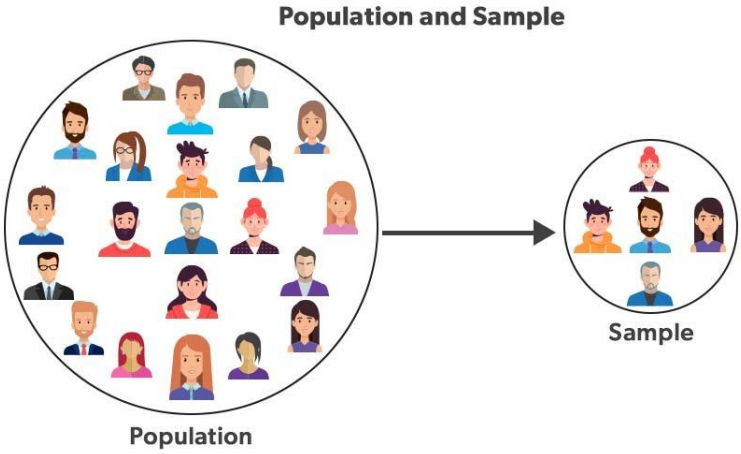
Data from the population is collected when a researcher or a business analyst needs a large amount of information about every person is available and easily accessible. population are used when the researcher question requires or you have access to data from every member of the population. Usually, the population is used when the datasets are quite small

example: In the university of 599 students if we want to remove the average BMI of each member of the population.

- ***How to collect data from the* sample*?***

Samples are used when the population is quite large in size or it is scattered or when it is impossible to collect data on the individual instances.

example: Let us assume the population of India is 10 million, and recent elections were conducted in India between two parties ‘party A ‘ and ‘party B’ now researchers want to find which party is winning so here we will create a group of few people lets say 10,000 from different regions and age groups so that sample is not biased. Then ask them who they voted we can get the exit poll. This is the thing which most of our media do during the elections, and show stats such as there 55% chances of party A winning the elections.


Population and Sample
Population
Sample

What Is Statistical Modeling?

tatistical modeling is like a formal depiction of a theory. It is typically described as the mathematical relationship between random and non-random variables.

The statistical modeling process is a way of applying statistical analysis to datasets in data science. The statistical model involves a mathematical relationship between random and non-random variables.

A statistical model can provide intuitive visualizations that aid data scientists in identifying relationships between variables and making predictions by applying statistical models to raw data.

Examples of common data sets for statistical analysis include census data, public health data, and social media data.

Reasons for learning statistical modeling

Even though data scientists are usually responsible for developing algorithms and models, analysts may also use statistical models in their work from time to time. As a result, analysts seeking to excel should gain a solid grasp of the factors that contribute to the success of these models.

Companies and organizations are leveraging statistical modeling to make predictions based on data to keep pace with the explosive growth of machine learning and artificial intelligence. The following are some benefits of understanding statistical modeling.

Machine learning vs. statistical modeling

Statistics and machine learning (ML) differ primarily in their purposes. You can build ML models for predicting the future by making accurate predictions without explicit programming, while statistical models can explain the relationship between variables.

However, some statistical models are inaccurate because of their inability to capture complex relationships between data, even if they can predict. ML

predictions are more accurate, but they are also more challenging to understand and explain.

In statistical models, probabilistic models for the data and variables are interpreted and identified, such as the effects of predictor variables. A statistical model establishes the magnitude and significance of relationships between variables and their scale. Models based on machine learning are more empirical.

What is Probability Distribution?

A Probability Distribution of a random variable is a list of all possible outcomes with corresponding probability values.

Note : The value of the probability always lies between 0 to 1.

What is an example of Probability Distribution?

Let’s understand the probability distribution by an example:

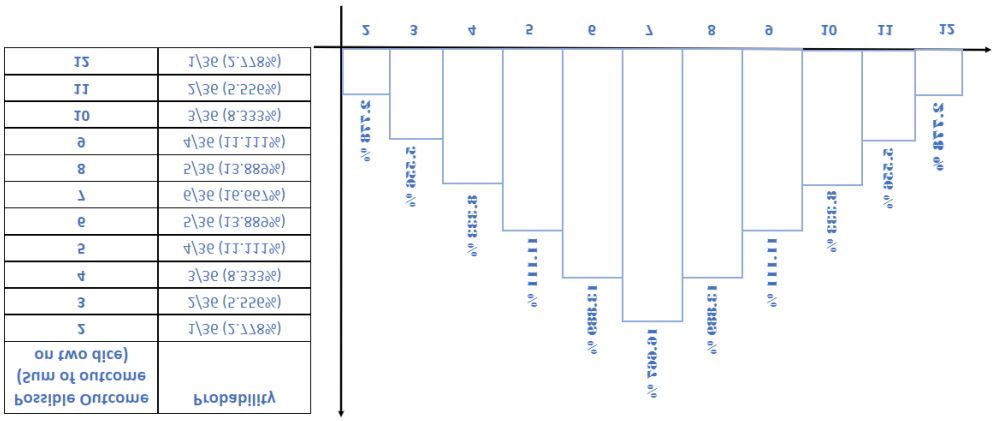
When two dice are rolled with six sided dots, let the possible outcome of rolling is denoted by (a, b), where

a : number on the top of first dice

b : number on the top of second dice

Then, sum of a + b are:

Sum of a + b	**(a, b)**
2	(1,1)
3	(1,2), (2,1)
4	(1,3), (2,2), (3,1)
5	(1,4), (2,3), (3,2), (4,1)
6	(1,5), (2,4), (3,3), (4,2), (5,1)
7	(1,6), (2,5), (3,4),(4,3), (5,2), (6,1)
8	(2,6), (3,5), (4,4), (5,3), (6,2)
9	(3,6), (4,5), (5,4), (6,3)
10	(4,6), (5,5), (6,4)



- If a random variable is a discrete variable, it's probability distribution is called discrete probability distribution.
  - Example : Flipping of two coins
  - Functions that represents a discrete probability distribution is known as **Probability Mass Function**.
- If a random variable is a continuous variable, it's probability distribution is called continuous probability distribution.
  - Example: Measuring temperature over a period of time
  - Functions that represents a continuous probability distribution is known as **Probability Density Function**.

Types of Probability Distributions

Uniform Distribution

What is Uniform Distribution?

Probability distribution in which all the outcome have equal probability is known as Uniform Distribution.

Example: Perfect Random Generator

What is an example of Uniform Distribution?

Let's understand by an example

Consider an experiment of tossing a single coin:

- **Probability of getting Head = 0.5**
- **Probability of getting Tail = 0.5**

- Random variable X is uniformly distributed if the distribution function is given by:

$$f(x) = \frac{1}{b-a},$$

where,

b : *heighest value of* X
a : *lowest value of* X

$$-\infty < a \leq x \leq b < \infty$$

Bernoulli Distribution

What is Bernoulli Distribution?

A discrete probability distribution for a random experiment that has only two possible outcomes (Bernoulli trials) is known Bernoulli Distribution.

Example: India will win cricket world cup or not

- It has only two possible outcome
  - Success (1)
  - Failure (0)
- Random variable n is Bernoulli distributed if the distribution function is given by:

$$P(n) = \begin{cases} 1-p & \text{for } n = 0 \\ p & \text{for } n = 1, \end{cases}$$

Where,

***p* = probability of success**

(1 - *p*) = q = probability of failure

What is an example of Bernoulli Distribution?

Let's understand by an example

Consider an experiment of Shooting of Basketball

- **Shoots the Ball *(n= 1) = p***
- **Doesn't Shoots the Ball *(n =0) =q = 1-p***

Binomial Distribution

What is Binomial Distribution?

A discrete probability distribution that gives only two possible outcomes in n independent trails is known as Binomial Distribution.

Example: Yes/No survey

- Extension of Bernoulli Distribution
- Represent the number of success and failure into n independent trials
- The probability of success and failure is the same for all independent and identical trails.
- Random variable X is binomial distributed if the distribution function is given by:

$$P(x: n, p) = \frac{n!}{[x!(n-x)!]} . p^x . q^{n-x}$$

Where,

n = number of trials (or number being sampled)

p = probability of success and

q = (1 - *p*) = probability of failure

- Mean = *np*
- Variance = *npq*
- *Mean > Variance*

What is an example of Binomial Distribution?

Let's understand the Binomial Distribution by an example,

Consider the experiment of Picking Balls

Problem Statement:

Let there are 8 white balls and 2 black balls, then the probability of drawing 3 white balls, if the probability of selecting white ball is 0.6.

$$n = 8 + 2 = 10$$

$$p = 0.6$$

$$P(X = 3) = \frac{10!}{3!7!}(0.6)^3(1 - 0.6)^7 = 0.04247$$

Difference between Binomial and Bernoulli's Distribution

Bernoulli	Binomial
Deals with the single trial event	Deals with the outcome of Multiple trials of the single events
Has only two possible outcome 0 and 1	Sum of identically and independent distributed Bernoulli Random Variable

Poisson Distribution

What is Poisson Distribution?

A discrete probability distribution that measures the probability of a random variable over a specific period of time is known as Poisson Distribution.

Example: Probability of Asteroid collision over a selected year of period.

- Used to predict probability of number of successful events.
- Random variable X is Poisson distributed if the distribution function is given by:

$$P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

where,

$$\lambda = average\ rate\ of\ the\ expected\ value$$

$$e(euler\ constant) = 2.718$$

Note: In case of Poisson Distribution **Mean = Variance**

What is an example of Poisson Distribution?

Let's understand the Poisson Distribution by an example,

Consider the experiment of Number of patient visiting in a hospital

Problem Statement :

Let in a hospital patient arriving in a hospital at expected value is 6, then what is the probability of five patients will visit the hospital in that day?

- **Patients arriving at expected value = 6**
- **P(Five patients will visit the hospital) = P(X=5)**

$$P(X=5) = \frac{6^5 e^{-6}}{5!} = 0.1606$$

Difference between Poisson Distribution and Binomial Distribution

Poisson	Binomial
Number of trials are infinite	Number of trials are fixed
Unlimited number of possible outcomes	Only two possible outcomes (Success or Failure)
Mean = Variance	Mean > Variance

Normal Distribution (Gaussian Distribution):

A continuous probability distribution, which is symmetric about it's mean value (i.e. data near the mean are more frequency in occurrence) is known as Normal Distribution.

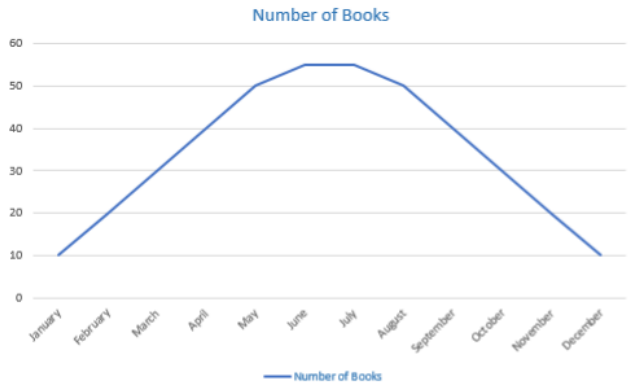
What is an example of Normal Distribution?

Lets' understand the Normal Distribution by an example,

Consider the experiment of Number of books read by students in a school

Number of Books Read by Students

Months	Number of Books
January	10
February	20
March	30
April	40
May	50
June	55
July	55
August	50
Septembe	40
October	30
Novembe	20
Decembe	10



- Random variable X is normally distributed if the distribution function is given by:

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

where

σ : *Standard Deviation*

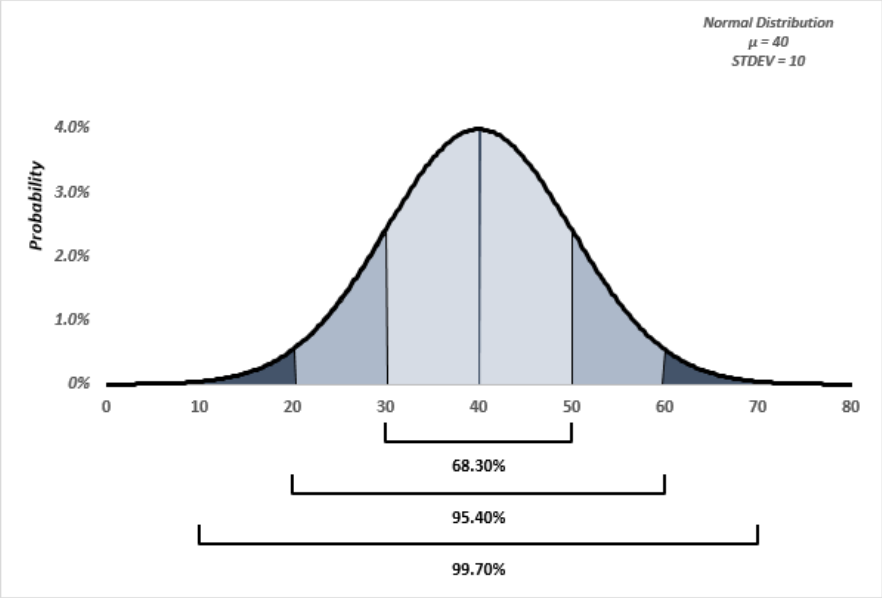
μ: *Mean*

x : *random variable*

Empirical Rule:

Empirical Rule is often called the ***68 – 95 – 99.7*** rule or **Three Sigma Rule**. It states that on a Normal Distribution:

- 68% of the data will be within one Standard Deviation of the Mean
- 95% of the data will be within two Standard Deviations of the Mean
- 99.7 of the data will be within three Standard Deviations of the Mean



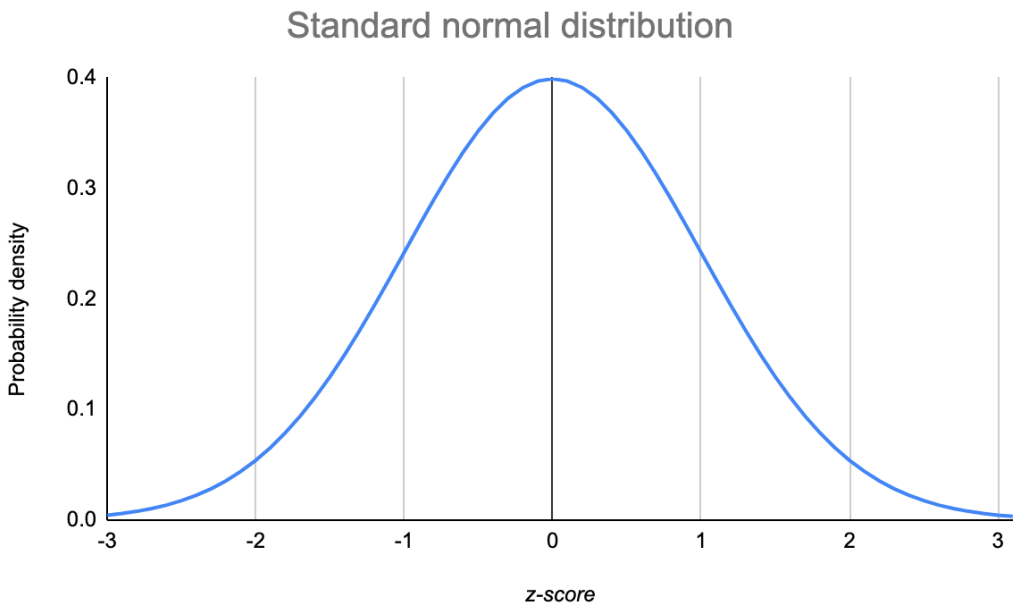
- Characteristics of Normal Distribution :
  - Symmetrical around its mean value

- Mean = Median = Mode
- Total area under the curve is 1
- Curve of the distribution is bell curve

Standard normal distribution

- Normal distribution with mean = 0 and standard deviation = 1.
- For any random Variable *X*, probability distribution function is given by:

$$f(X = x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, \quad -\infty < x < \infty$$



Probability distributions are not a Graph.

A graph is just a visual representation

Difference between Poisson and Normal Distribution	
Poisson	**Normal**
Use Discrete Data	Use Continuous Data
Distribution vary on mean value	Symmetric about mean value
Mean = Variance	Mean = Median = Mode

Model Fitting

Chapter 3 - Model Fitting

Summary: Laws look like continuous sequences of points in data, but models look like noisy clouds of points. As a result, it is much harder to reverse engineer a model from data than it is to reverse engineer a law. How should we pick the model that best describes the data? We can use inverse probability, or likelihood to find the model that is *most likely* to have generated the data. This allows us to fit many different types of models. Linear. Linear discrete. Linear multivariate. Linear multivariate with interaction terms. Generalized linear models. And non-linear Generalized Additive Models.

---

1. How to fit?

1. Both laws and models appear as patterns in data. However it is much easier to spot a law with these patterns than it is to spot a model.
    1. Why should this be? For a law, each data point falls *on* the function that describes the law.
    2. But for a model, each data point falls *around* the function that describes the model. In fact, the data points are *distributed* around the function (as you know).
        1. With enough points, you may be able to notice the structure of these distributions, and therefore make a good guess about the form of the model, but you will rarely have this many points.
        2. More commonly you will need to make the best guess that you can about the model given the data that you have.
        3. In practice, you use an algorithm to select the best guess and R takes care of the details
            1. `lm()`
            2. `xyplot()`, `makeFun()`, `plotFun(add = TRUE)`
        4. But what is R doing?
2. Many algorithms exist to help you spot the "best" model given the data. In this chapter we will look at how the most popular algorithms work and learn to use them. In the following chapter, we will consider how to choose between the algorithms when modeling a particular data set.

---

2. Likelihood, the inverse of probability

1. The most intuitive modeling algorithms rely on likelihood. In short, they pick the model that is most *likely* to have generated the data.
2. We use the term likely in everyday speech, but in science *likelihood* has a specific meaning that is closely related to probability.
    1. Probability describes the chance that a certain observation will come to pass given a model. It reasons from model to (future) observation.
    2. Likelihood describes the chance that a given model caused the observation that has come to pass. It reasons from observation to model. For this reason, likelihood is sometimes called inverse probability.

        1. Although likelihood and probability both deal with chances they behave differently mathematically. Most notably, the probability of disjoint events must sum to one. Not so for likelihoods.
3. You can calculate the likelihood that a specific model produced a set of data.
    1. Example of a very simple model
        1. `goal(y ~ 1, data = ...)`
        2. `lm()`
    2. To find the most likely model compare the likelihoods of different models and choose the model with the highest likelihood.
    3. There is no guarantee that the model with the highest likelihood generated the data, but it is the most pragmatic conclusion to draw. This method of reasoning is known as abduction.
4. In practice, there is a logistical issue to address. There is an infinite number of models to choose from. How will you ever stop calculating likelihoods?
    1. Well, the same way you manage possibilities whenever you make a decision in real life. You first narrow the possibilities down to a smaller set for further consideration. Each modeling algorithm only applies to a small set of models, the set of models that have the structure the algorithm optimizes likelihoods over.
        1. By choosing to use an algorithm, you narrow the set of models to consider to a tractable number. The algorithm then identifies the most likely model within that set.
        2. You should choose the algorithm that matches the structure of the natural law that you wish to model (if you know this structure).
        3. In the next chapter, we will consider how to proceed when you do not know anything about this structure.

---

3. Fitting a linear model

1. Linear models are among the oldest and most interpretable modeling methods. A linear model uses a linear function to map a set of values to a set of normal distributions.
    1. Linear models are widely useful because
        1. The normal distribution occurs frequently in the natural world.
        2. Any continuous function can be approximated well with a straight line over a short distance.
    2. Linear models also have another useful feature. For linear models (and generalized linear models with exponential distributions), the model with maximum likelihood will be the model that has the smallest residual sum of squares.
        1. A residual is the distance between the mean of the distribution predicted by a model and the actual data point observed.
        2. Notice that residuals can be positive or negative. Squaring the residuals is a way to measure the magnitude of a residual.
        3. Hence the least squares model will be the model that has the smallest squared residuals, i.e. the model that comes closest to the data points.
2. The `lm()` function fits a linear model to data.
    1. `goal(y ~ x, data = ...)`
    2. `lm()`
3. R's modeling functions return an object that contains a lot of data. To access the data, *store then explore*.
    1. `resid()`
    2. `coef()`
    3. `fitted()`

    4. `xyplot()`, `makeFun()`, `plotFun(add = TRUE)`
4. Linear models are particularly easy to interpret. The coefficient of X is the number of units that the best guess of Y ($\hat{Y}$) increases as X increases by one unit.

---

4. Discrete terms

1. You can also apply linear models to discrete terms.
    1. Consider this example. To explore the data:
        1. `diffmean()`
        2. `tally()`
        3. `prop()`
        4. `perc()`
    2. Build your model as you normally would
        1. `goal(y ~ 1 | z, data = ...)`
        2. `lm()`
2. Interpretation
    1. R will provide a coefficient for each level of the discrete variable except one. This level will be used as a baseline.
    2. The intercept is the best guess of Y for the baseline group
    3. Each β coefficient is a modifier. It shows how to modify the baseline coefficient to determine the best guess of Y for each remaining group. In other words, it shows the change in the best guess of Y that results from switching from the baseline group to the coefficient's group.
    4. Use `factor()` to change the baseline group. Your coefficients will change, but the final results will not.

---

5. Multivariate models

1. Add additional terms to the formula to include additional predictors in your model
    1. `goal(y ~ x | z, data = ...)`
    2. `goal(y ~ x + z, data = ...)`
2. Now each coefficient should be interpreted as the change in the best guess of Y that results from changing X_i by one unit *while holding the values of the other* *X_j constant*.

---

6. Interaction terms

1. Multivariate models create the possibility of interaction effects. An interaction effect occurs when the values of one variable modify the effect of another variable.
    1. A visual explanation
2. Notation
    1. `goal(y ~ x + z + x:z, data = ...)`
    2. `goal(y ~ x*z, data = ...)`
3. Interpretation

---

7. Generalized models

1. `lm()` uses likelihood to find the model that maps values of *X* to distributions in *Y* that have a *normal* distribution. We saw in Chapter 2 that many natural events will have a normal distribution, but other will not. What if we want to use a model that maps values to non-normal distributions? (This would be appropriate if we are modeling a non-normal *Y* or even a discrete *Y*)
    1. This is an important change because the distributions associated with a model determine how well it fits. They determine how likely the model is to generate the data.
2. It is easy to generalize linear models to non-normal cases by modeling a function of *Y*.
    1. Compare linear model equation to glm model equation
    2. Such functions are known as *link functions*. They map non-normal input (*Y*) to normal output, which can be fitted in the usual way.
    3. These models are known as *generalized linear models (GLM)*
    4. The most common form of generalized linear models are logisitic models
    5. `glm(..., family = ...)`
        1. `options(na.action = "na.exclude")`
        2. `predict(mod, type = "link"); predict(mod, type = "link")`
        3. `resid(gmod, type = "deviance"); resid(gmod, type = "pearson")`
3. To interpret a generalized linear model, back transform the coefficients through the link function, or simply try to understand the predictions
    1. Interpreting a logistic regression

---

8. Non-linear models

1. We can use a similar strategy to model non-linear relationships. Instead of modeling *Y* on *X*, we can model *Y* on a function of *X*.
    1. If you have a particular function of *X* in mind, you can put it straight into the model equation.
        1. `lm()`
        2. Don't forget to back transform your results when interpreting
2. *General additive models (GAMS)* fit a model that maps smooth functions of the X_i to distributions in *Y*. These functions do not need to be linear, only smooth, which makes GAM algorithms useful for mapping many types of relationships.
    1. GAM model equation
    2. `library(mgcv)`
    3. `gam()`
    4. `s()`
        1. interactions
3. As before, we are using likelihood to identify the best model, but our low level tactics have changed. The model with the highest likelihood will be the model whose *X* functions put the model line exactly through each data point, something that may now be feasible. However, such a model is as unlikely to be true in the practical sense as it is likely to be true in the mathematical sense. As a result, `gam()` uses a penalized iterative method to select the most likely sensible model.
4. You can combine generalized linear methods and generalized additive methods with `gam()`.
    1. Model equation
    2. Hence, you can think of `gam()` as a type of generalized modeling algorithm

5. Interpreting the results of GAMs is difficult. For GAMs as well as other modeling methods that we will encounter later, interpret the model through its predictions.

---

9. Summary

1. At its heart, model fitting is an optimization algorithm. Each of the methods above optimizes a likelihood function to find the "best fitting" model.
    1. Recommended reading for the mathematics behind model fitting: The Elements of Statistical Learning
2. Each of these methods finds the best *parametric* model to fit your data. It is hard to describe a model (which must describe all possible data points) without using a parametric distribution.
    1. We will look at some *non-parametric* models in Chapter 6.
3. This chapter covered the most popular (and the most accessible) methods of model fitting, but many more modeling algorithms exist.
    1. Additional reference for other methods: Applied Predictive modeling
    2. How do you know which method you should use with your data? Chapter 4 will begin with this question.

What is Exploratory Data Analysis ?

Exploratory Data Analysis (EDA) is an approach to analyze the data using visual techniques. It is used to discover trends, patterns, or to check assumptions with the help of statistical summary and graphical representations.

Exploratory data analysis Types

TYPES OF EXPLORATORY DATA ANALYSIS:

1. Univariate Non-graphical
2. Multivariate Non-graphical
3. Univariate graphical
4. Multivariate graphical

1. Univariate Non-graphical: this is the simplest form of data analysis as during this we use just one variable to research the info. The standard goal of univariate non-graphical EDA is to know the underlying sample distribution/ data and make observations about the population. Outlier detection is additionally part of the analysis. The characteristics of population distribution include:

- **Central tendency:** The central tendency or location of distribution has got to do with typical or middle values. The commonly useful measures of central tendency are statistics called mean, median, and sometimes mode during which the foremost common is mean. For skewed distribution or when there's concern about outliers, the median may be preferred.
- **Spread:** Spread is an indicator of what proportion distant from the middle we are to seek out the find the info values. the quality deviation and variance are two useful measures of spread. The variance is that the mean of the square of the individual deviations and therefore the variance is the root of the variance
- **Skewness and kurtosis:** Two more useful univariates descriptors are the skewness and kurtosis of the distribution. Skewness is that the measure of asymmetry and kurtosis may be a more subtle measure of peakedness compared to a normal distribution

2. Multivariate Non-graphical: Multivariate non-graphical EDA technique is usually wont to show the connection between two or more variables within the sort of either cross-tabulation or statistics.

- For categorical data, an extension of tabulation called cross-tabulation is extremely useful. For 2 variables, cross-tabulation is preferred by making a two-way table with column headings that match the amount of one-variable and row headings that match the amount of the opposite two variables, then filling the counts with all subjects that share an equivalent pair of levels.
- For each categorical variable and one quantitative variable, we create statistics for quantitative variables separately for every level of the specific

variable then compare the statistics across the amount of categorical variable.
- Comparing the means is an off-the-cuff version of ANOVA and comparing medians may be a robust version of one-way ANOVA.

3. Univariate graphical: Non-graphical methods are quantitative and objective, they are not able to give the complete picture of the data; therefore, graphical methods are used more as they involve a degree of subjective analysis, also are required. Common sorts of univariate graphics are:

- **Histogram:** The foremost basic graph is a histogram, which may be a barplot during which each bar represents the frequency (count) or proportion (count/total count) of cases for a variety of values. Histograms are one of the simplest ways to quickly learn a lot about your data, including central tendency, spread, modality, shape and outliers.
- **Stem-and-leaf plots:** An easy substitute for a histogram may be stem-and-leaf plots. It shows all data values and therefore the shape of the distribution.
- **Boxplots:** Another very useful univariate graphical technique is that the boxplot. Boxplots are excellent at presenting information about central tendency and show robust measures of location and spread also as providing information about symmetry and outliers, although they will be misleading about aspects like multimodality. One among the simplest uses of boxplots is within the sort of side-by-side boxplots.
- **Quantile-normal plots:** The ultimate univariate graphical EDA technique is that the most intricate. it's called the quantile-normal or QN plot or more generally the quantile-quantile or QQ plot. it's wont to see how well a specific sample follows a specific theoretical distribution. It allows detection of non-normality and diagnosis of skewness and kurtosis

4. Multivariate graphical: Multivariate graphical data uses graphics to display relationships between two or more sets of knowledge. The sole one used commonly may be a grouped barplot with each group representing one level of 1 of the variables and every bar within a gaggle representing the amount of the opposite variable.

The Other common types of multivariate graphics include:

- **Scatter plot:** Is used to plot data points on a horizontal and a vertical axis to show how much one variable is affected by another.
- **Multivariate chart:** Is a graphical representation of the relationships between factors and a response.
- **Run chart:** Is a line graph of data plotted over time.
- **Bubble chart:** Is a data visualization that displays multiple circles (bubbles) in a two-dimensional plot.

- **Heat map:** Is a graphical representation of data where values are depicted by color.

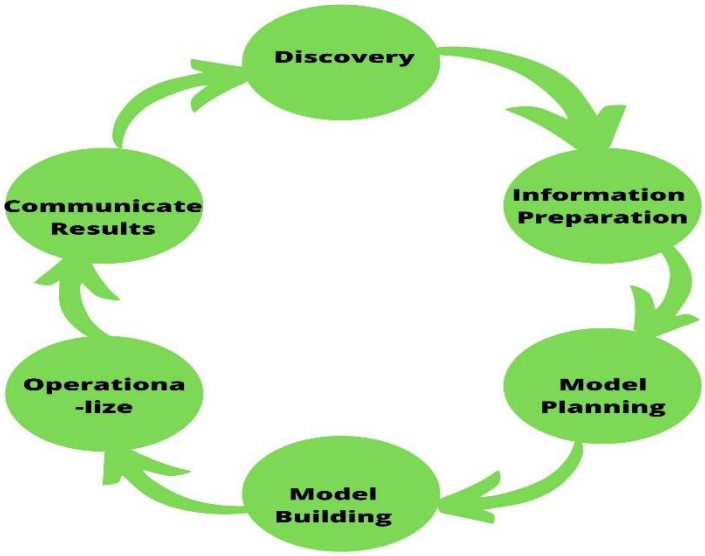
Data Science Process

Data Science could be a space that incorporates working with colossal sums of information, creating calculations, working with machine learning and more to come up with trade insights. It incorporates working with the gigantic sum of information. Different processes are included to infer the information from the source like extraction of data, information preparation, model planning, model building and many more. The below image depicts the various processes of Data Science.

Let's go through each process briefly.

- **Discovery** To begin with, it is exceptionally imperative to get the different determinations, prerequisites, needs and required budget-related with the venture. You must have the capacity to inquire the correct questions like do you have got the desired assets. These assets can be in terms of individuals, innovation, time and information. In this stage, you too got to outline the trade issue and define starting hypotheses (IH) to test.
- **Information Preparation** In this stage, you would like to investigate, preprocess and condition data for modeling. You'll be able to perform information cleaning, changing, and visualization. This will assist you to spot the exceptions and build up a relationship between the factors. Once you have got cleaned and arranged the information, it's time to do exploratory analytics on it.
- **Model Planning** Here, you may decide the strategies and methods to draw the connections between factors. These connections will set the base for the calculations which you may execute within the following stage. You may apply Exploratory Data Analytics (EDA) utilizing different factual equations and visualization apparatuses.
- **Model Building** In this stage, you'll create datasets for training and testing purposes. You may analyze different learning procedures like classification, association, and clustering and at last, actualize the most excellent fit technique to construct the show.
- **Operationalize** In this stage, you convey the last briefings, code, and specialized reports. In expansion, now a pilot venture is additionally actualized in a real-time generation environment. This will give you a clear picture of the execution and other related limitations.

- **Communicate Results** Presently, it is critical to assess the outcome of the objective. So, within the final stage, you recognize all the key discoveries, communicate to the partners and decide in the event that the outcomes about the venture are a victory or a disappointment based on the criteria created in Stage



Use of Data Science Process :

The Data Science Process is a systematic approach to solving data-related problems and consists of the following steps:

1. Problem Definition: Clearly defining the problem and identifying the goal of the analysis.

2. Data Collection: Gathering and acquiring data from various sources, including data cleaning and preparation.

3. Data Exploration: Exploring the data to gain insights and identify trends, patterns, and relationships.

4. Data Modeling: Building mathematical models and algorithms to solve the problem and make predictions.

5. Evaluation: Evaluating the model's performance and accuracy using

appropriate metrics.

6. Deployment: Deploying the model in a production environment to make predictions or automate decision-making processes.

Monitoring and Maintenance: Monitoring the model's performance over time and making updates as needed to improve accuracy.

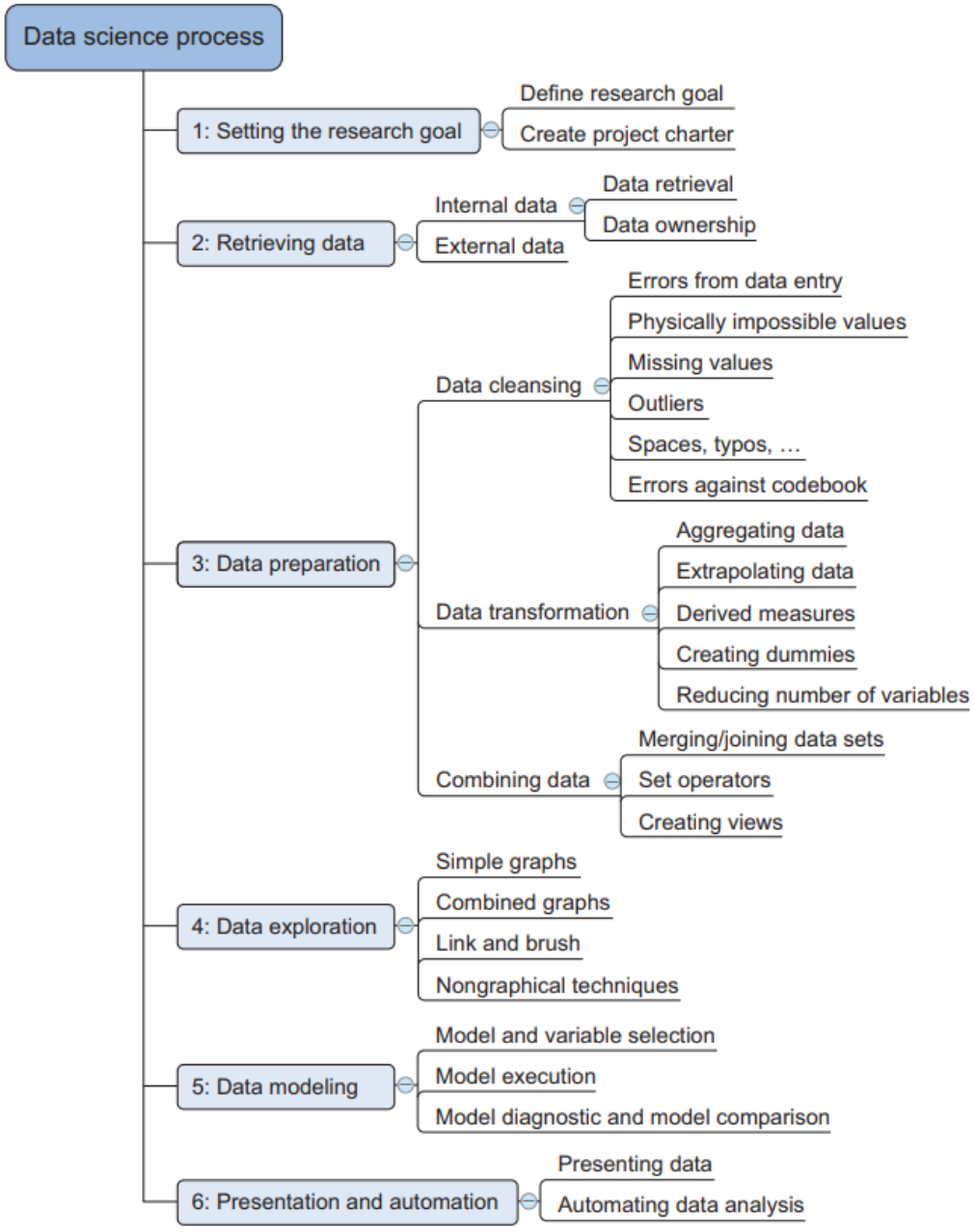


Figure 2.1 The six steps of the data science process

An introduction to Machine Learning

Definition of Machine Learning: Arthur Samuel, an early American leader in the field of computer gaming and artificial intelligence, coined the term “Machine Learning ” in 1959 while at IBM. He defined machine learning as “the field of study that gives computers the ability to learn without being explicitly programmed “. However, there is no universally accepted definition for machine learning. Different authors define the term differently. We give below two more definitions.

- Machine learning is programming computers to optimize a performance criterion using example data or past experience . We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be predictive to make predictions in the future, or descriptive to gain knowledge from data.
- The field of study known as machine learning is concerned with the question of how to construct computer programs that automatically improve with experience.
    - Machine learning is a subfield of artificial intelligence that involves the development of algorithms and statistical models that enable computers to improve their performance in tasks through experience. These algorithms and models are designed to learn from data and make predictions or decisions without explicit instructions. There are several types of machine learning, including supervised learning, unsupervised learning, and reinforcement learning. Supervised learning involves training a model on labeled data, while unsupervised learning involves training a model on unlabeled data. Reinforcement learning involves training a model through trial and error. Machine learning is used in a wide variety of applications, including image and speech recognition, natural language processing, and recommender systems.
    - **Definition of learning:** A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks T, as measured by P , improves with experience E.
    - **Examples**
    - Handwriting recognition learning problem
        - Task T : Recognizing and classifying handwritten words within images
        - Performance P : Percent of words correctly classified
        - Training experience E : A dataset of handwritten words with given classifications

- A robot driving learning problem
  - Task T : Driving on highways using vision sensors
  - Performance P : Average distance traveled before an error
  - Training experience E : A sequence of images and steering commands recorded while observing a human driver

Definition: A computer program which learns from experience is called a machine learning program or simply a learning program .

Classification of Machine Learning

Machine learning implementations are classified into four major categories, depending on the nature of the learning "signal" or "response" available to a learning system which are as follows:

A. Supervised learning:

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. The given data is labeled . Both *classification* and *regression* problems are supervised learning problems .

- Example — Consider the following data regarding patients entering a clinic . The data consists of the gender and age of the patients and each patient is labeled as "healthy" or "sick".

gender	age	label
M	48	sick
M	67	sick
F	53	healthy
M	49	sick
F	32	healthy
M	34	healthy
M	21	healthy

B. Unsupervised learning:

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses. In unsupervised learning algorithms, classification or categorization is not included in the observations. Example: Consider the following data regarding patients entering a clinic. The data consists of the gender and age of the patients.

gender	age
M	48
M	67
F	53
M	49
F	34
M	21

As a kind of learning, it resembles the methods humans use to figure out that certain objects or events are from the same class, such as by observing the degree of similarity between objects. Some recommendation systems that you find on the web in the form of marketing automation are based on this type of learning.

C. Reinforcement learning:

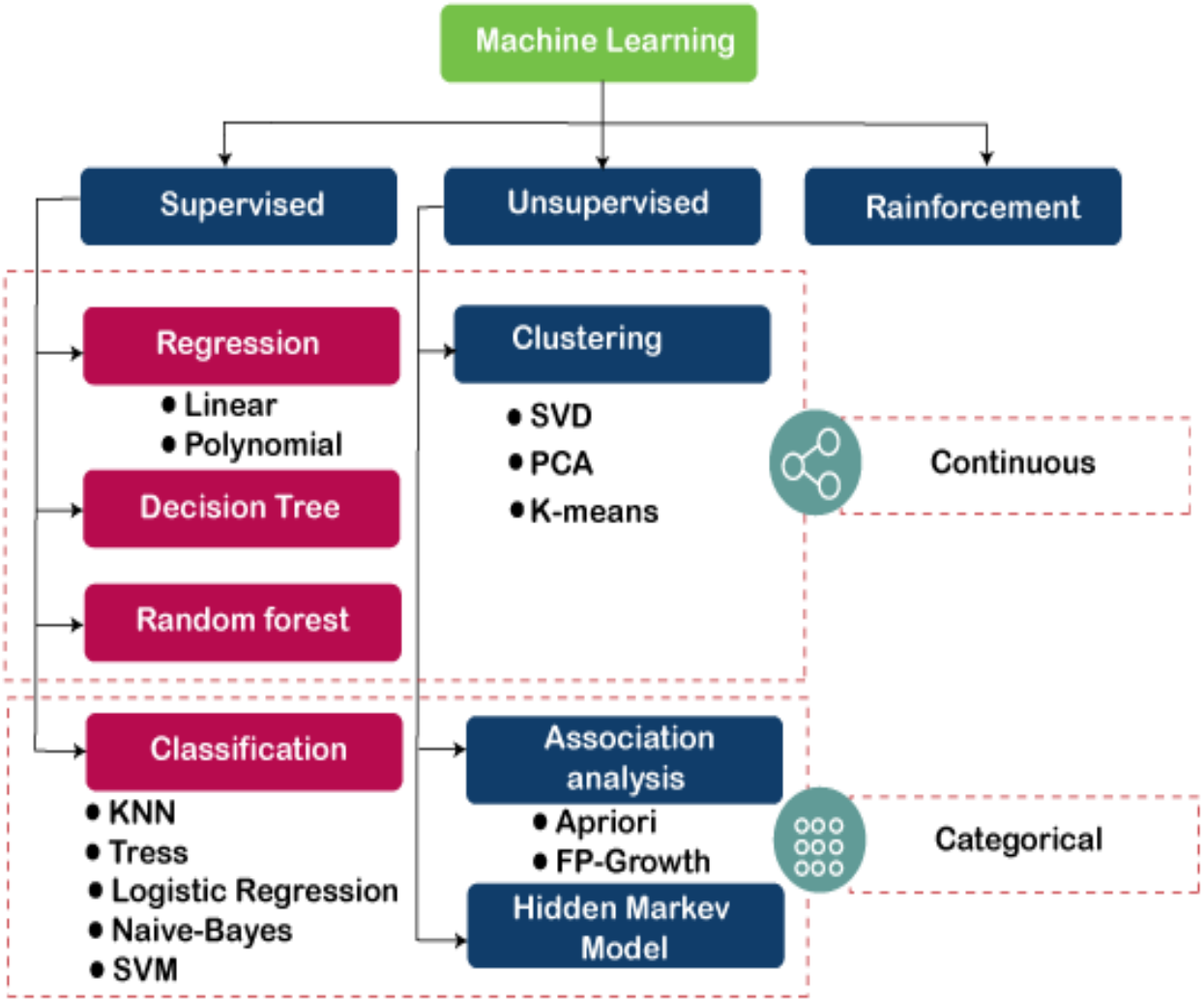
Reinforcement learning is the problem of getting an agent to act in the world so as to maximize its rewards.

A learner is not told what actions to take as in most forms of machine learning but instead must discover which actions yield the most reward by trying them. For example — Consider teaching a dog a new trick: we cannot tell him what to do, what not to do, but we can reward/punish it if it does the right/wrong thing.

When watching the video, notice how the program is initially clumsy and unskilled but steadily improves with training until it becomes a champion.

D. Semi-supervised learning:

Where an incomplete training signal is given: a training set with some (often many) of the target outputs missing. There is a special case of this principle known as Transduction where the entire set of problem instances is known at learning time, except that part of the targets are missing. Semi-supervised learning is an approach to machine learning that combines small labeled data with a large amount of unlabeled data during training. Semi-supervised learning falls between unsupervised learning and supervised learning.



List of Popular Machine Learning Algorithm

1. **Linear Regression Algorithm**
2. **Logistic Regression Algorithm**
3. **Decision Tree**
4. **SVM**
5. **Naïve Bayes**
6. **KNN**
7. **K-Means Clustering**
8. **Random Forest**
9. **Apriori**
10. **PCA**

1. Linear Regression

Linear regression is one of the most popular and simple machine learning algorithms that is used for predictive analysis. Here, **predictive analysis** defines prediction of something, and linear regression makes predictions for *continuous numbers* such as **salary, age, etc.**

It shows the linear relationship between the dependent and independent variables, and shows how the dependent variable(y) changes according to the independent variable (x).

It tries to best fit a line between the dependent and independent variables, and this best fit line is knowns as the regression line.

The equation for the regression line is:

$$y= a_0+ a*x+ b$$

Here, y= dependent variable

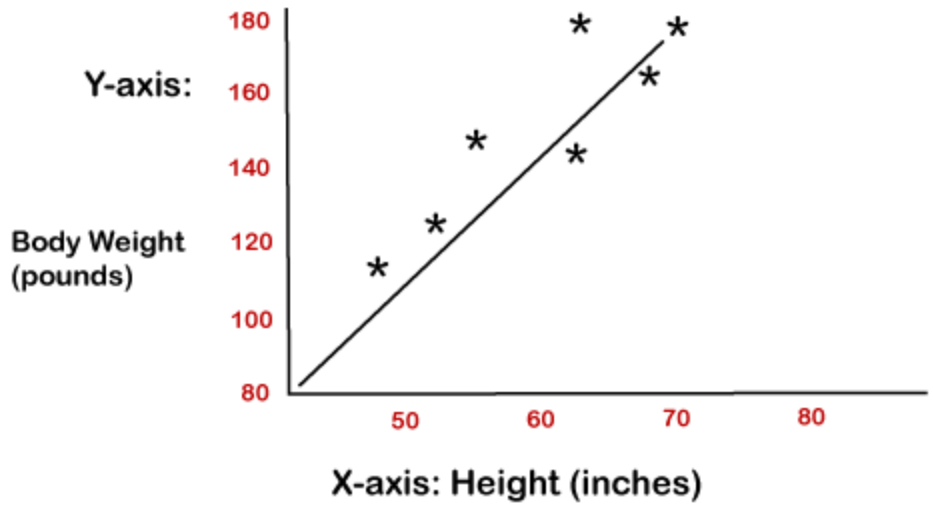
x= independent variable

a_0 = Intercept of line.

Linear regression is further divided into two types:

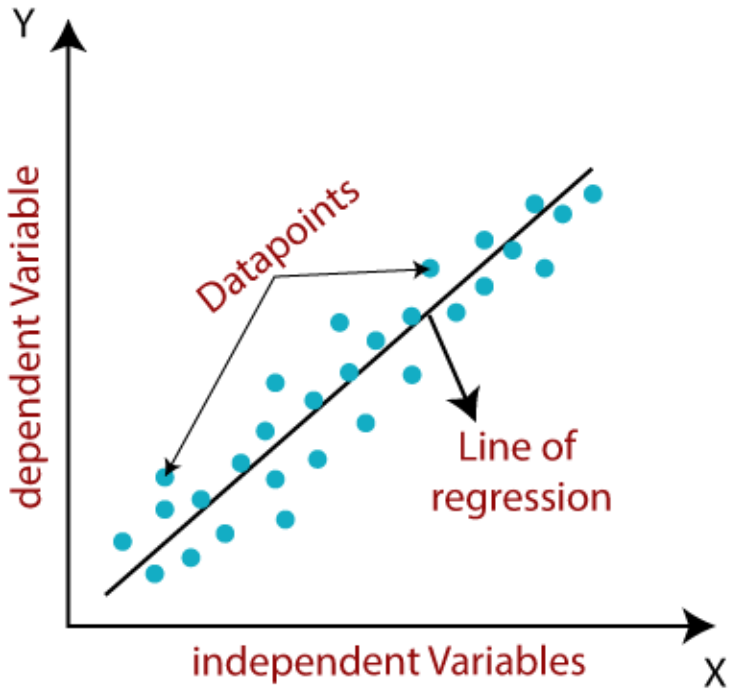
- **Simple Linear Regression:** In simple linear regression, a single independent variable is used to predict the value of the dependent variable.
- **Multiple Linear Regression:** In multiple linear regression, more than one independent variables are used to predict the value of the dependent variable.

The below diagram shows the linear regression for prediction of weight according to height



Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



$$y= a_0+a_1x+ \varepsilon$$

Here,

Y=DependentVariable(TargetVariable)
X=IndependentVariable(predictorVariable)
a0=interceptoftheline (Gives an additional degree of freedom)
a1 = Linear regression coefficient (scale factor to each input value).
ε = random error

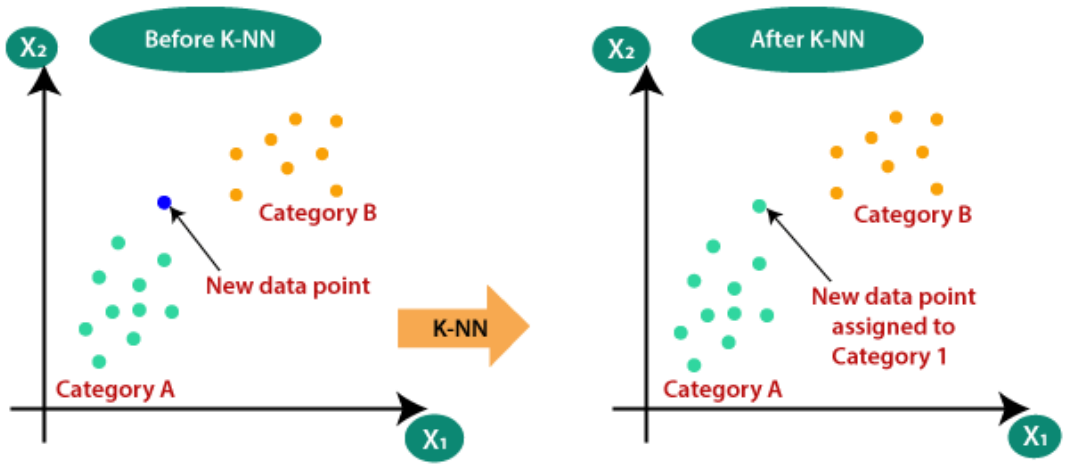
The values for x and y variables are training datasets for Linear Regression model representation.

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

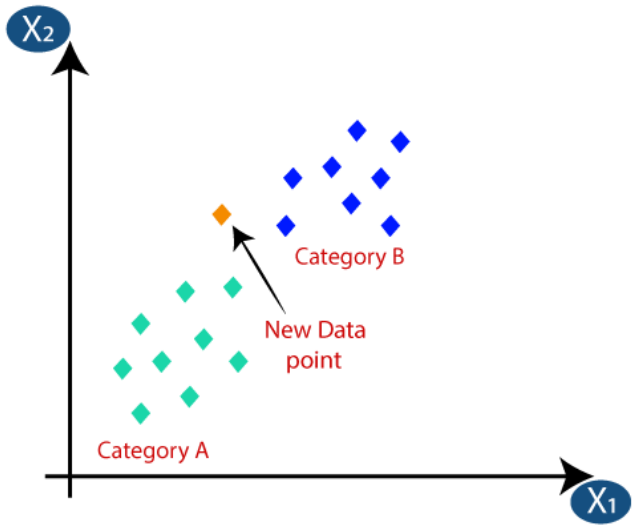


How does K-NN work?

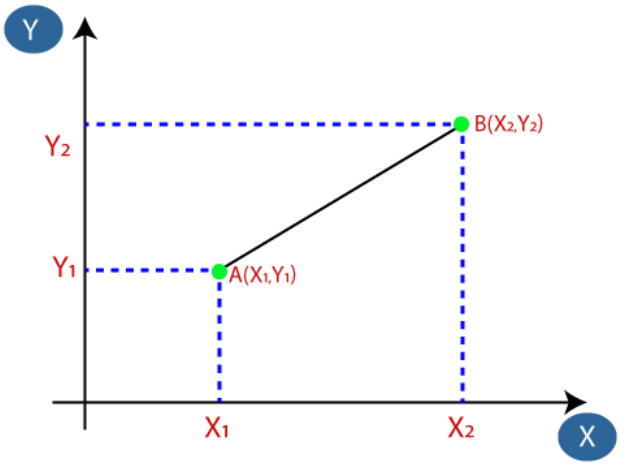
The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

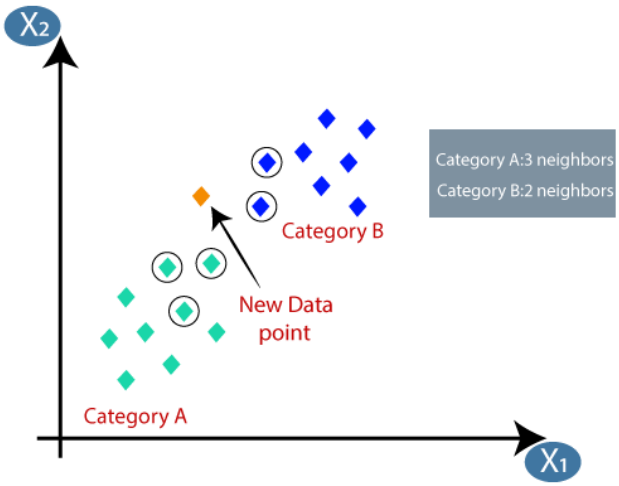


- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



Euclidean Distance between A_1 and $B_2 = \sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

What *K* in *K*-NN stands for?

K in *K*-Nearest Neighbors refers to the number of neighbors that one should take into consideration when predicting (voting for) the class of a new point. It will get more clear from the below example.

K-Means

K-Means (aka *K*-Means clustering) is an unsupervised learning algorithm that divide unlabeled data into different groups (or clusters). *K* in *K*-means refers to the number of clusters/groups (a cluster is a group of similar observations/records). For instance, in the following example, the unlabeled dataset is grouped into different number of clusters depending on the value of *K*.

K-Means minimizes the *within-cluster sum of squares*, **WCSS** (aka intracluster inertia/distance, within-cluster variance). To put it simply, K-Means minimizes the sum of squared differences between data points and the mean of the assigned cluster

$$\text{WCSS}_k = \sum_{x \in k} ||x - x||_2$$

How to find the best *K*?

There are several ways to determine *K* in the *K*-Means clustering algorithm:

- **Elbow Method**: A common way to determine the number of ideal cluster (K) in *K*-means. In this approach, we run the *K*-means with several candidates and calculate the WCSS. The best *K* is selected based on a trade-off between the model complexity (overfitting) and the WCSS.
- **Silhouette Score**: A score between -1 and 1 measuring the similarity among points of a cluster and comparing that with other clusters. A score of -1 indicates that a point is in the wrong cluster, whereas a score of 1 indicates that the point is in the right cluster **gap statistics**: A method to estimate the number of clusters in a dataset. gap statistic compares the change in the within-cluster variation of output of any clustering technique with an expected reference null distribution

We usually normalize/standardize continuous variables in the data preprocessing stage in order to avoid variables with much larger values dominating any modeling or analysis process

Pros and Cons

Some pros and cons of *K*-Means are given below.

Pros

- High scalability since most of calculations can be run in parallel.

Cons

- The outliers can skew the centroids of clusters.
- Poor performance in higher dimensional.

Conclusion

Few takeaways from this post:

- *K*NN is a supervised learning algorithm mainly used for classification problems, whereas *K*-Means (aka *K*-means clustering) is an unsupervised learning algorithm.
- *K* in *K*-Means refers to the number of clusters, whereas *K* in *K*NN is the number of nearest neighbors (based on the chosen distance metric).
- *K* in *K*NN is determined by comparing the performance of algorithm using different values for *K*.
- There are few ways to determine the number of groups/clusters in a dataset prior to the *K*-means clustering, and that are:
    - Elbow Method,
    - Silhouette Score,
    - gap statistic.

Applications of Machine learning

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

1. Image Recognition:

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

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

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

2. Speech Recognition

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

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

3. Traffic prediction:

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

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

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

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

4. Product recommendations:

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

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

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

5. Self-driving cars:

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

6. Email Spam and Malware Filtering:

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

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

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

7. Virtual Personal Assistant:

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

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

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

8. Online Fraud Detection:

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

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

9. Stock Market trading:

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

10. Medical Diagnosis:

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

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

11. Automatic Language Translation:

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

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

Naive Bayes: Spam or Ham?

Nobody likes spam emails. How can Bayes help? In this chapter, we'll keep expanding our data science knowledge with a practical example. A simple yet effective way of using Bayesian probability to create a spam filter from scratch will be introduced. The filter will examine emails and classify them as either spam or ham (the word for non-spam emails) based on their content. What we will be implementing here is a *supervised learning model*, in other words, a classification model that has been trained on previously classified data. Think of it like a machine to which you can give some input, like an email, and will give you some label to that input, like spam or ham. This machine has a lot of tiny knobs, and based on their particular configuration it will output some label for each input. Supervised learning involves iteratively finding the right configuration of these knobs by letting the machine make a guess with some pre-classified data, checking if the guess matches the true label, and if not, tune the knobs in some controlled way. The way our machine will make predictions is based on the underlying mathematical model. For a spam filter, a *naive Bayes* approach has proven to be effective, and you would have the opportunity to verify that yourself at the end of the chapter. In a naive Bayes model, Bayes' theorem is the main tool for classifing, and it is *naive* because we make very loose assumptions about the data we are analyzing. This will be clearer once we dive into the implementation.

The term spam has its origins in the 1970s, in a sketch from the British comedy troupe Monty Python (who also inspired the name of the Python programming language). In this sketch from the series *Monty Python's Flying Circus*, a customer wants to order food at a restaurant, but as the waitress begins to loudly recite the menu, the word *spam* starts to appear with increasing frequency, to the point of absurdity. Eventually all meaning is lost, and the waitress is just shouting *spam, spam, spam* . . . which is often what an email inbox feels like.

4.2 The Training Data

For the Bayesian spam filter to work correctly, we need to feed it some good training data. In this context, that means having a large enough corpus of emails that have been pre-classified as spam or ham. The emails should be collected from a sufficiently heterogeneous group of people. After all, spam is a somewhat subjective category: one person's spam may be another person's ham. The proportion of spam vs. ham in our data should also be somewhat representative of the real proportion of emails we receive.

Fortunately, there are a lot of very good datasets available online. We'll use the "Email Spam Classification Dataset CSV" from Kaggle a website where data science enthusiasts and practitioners publish datasets, participate in competitions, and share their knowledge. The dataset's description included online helps us make sense of its contents:

The .csv file contains 5,172 rows, one row for each email. There are 3,002 columns. The first column indicates Email name. The name has been set with numbers and not recipients' name to protect privacy. The last column has the labels for prediction: for spam, for not spam. The remaining 3,000 columns are the 3,000 most common words in all the emails, after excluding the non-alphabetical characters/words. For each row, the count of each word(column) in that email(row) is stored in the respective cells.

Let's take a look at the data. The following code snippet outputs a view of the first and last rows of the dataset.

```
raw_df = CSV.read("./04_naive_bayes/data/emails.csv", DataFrame)
## 5172×3002 DataFrame
##   Row │ Email No.   the    to     ect    and    for    of     a      you    ho
…
##       │ String15    Int64  Int64  Int64  Int64  Int64  Int64  Int64  Int64  In
…
##
──────┼──────────────────────────────────────────────────────────────────────────
##     1 │ Email 1         0      0      1      0      0      0      2      0
…
##     2 │ Email 2         8     13     24      6      6      2    102      1
##     3 │ Email 3         0      0      1      0      0      0      8      0
##     4 │ Email 4         0      5     22      0      5      1     51      2
##     5 │ Email 5         7      6     17      1      5      2     57      0
…
##     6 │ Email 6         4      5      1      4      2      3     45      1
##     7 │ Email 7         5      3      1      3      2      1     37      0
##     8 │ Email 8         0      2      2      3      1      2     21      6
##   ⋮   │     ⋮         ⋮      ⋮      ⋮      ⋮      ⋮      ⋮      ⋮      ⋮      ⋱
##  5166 │ Email 5166      1      0      1      0      3      1     12      1
…
##  5167 │ Email 5167      1      0      1      1      0      0      4      0
##  5168 │ Email 5168      2      2      2      3      0      0     32      0
##  5169 │ Email 5169     35     27     11      2      6      5    151      4
##  5170 │ Email 5170      0      0      1      1      0      0     11      0
…
##  5171 │ Email 5171      2      7      1      0      2      1     28      2
##  5172 │ Email 5172     22     24      5      1      6      5    148      8
##                                                  2993 columns and 5157 rows
omitted
```

As you can see, the output informs the amount of rows and columns and the type of each column and allows us to see a sample of the data.

4.3 Preprocessing the Data

Before we use the data to train our filter, we need to preprocess it a little bit. First, we should filter out very common words, such as articles and pronouns, which will most likely add noise rather than information to our classification algorithm.

```
# Get the list of all words present in all mails.
# We strip the first and last columns which are the email id and label
```

```
all_words = names(raw_df)[2:end-1]

# Create a StringDocument, a struct with methods to remove articles and
# pronouns from the text
all_words_text = join(all_words, " ")
document = StringDocument(all_words_text)

# Remove articles and pronouns
prepare!(document, strip_articles)
prepare!(document, strip_pronouns)

# Create another DataFrame with the filtered words.
vocabulary = split(TextAnalysis.text(document))
clean_words_df = raw_df[!, vocabulary]

# Transform the DataFrame into a Matrix and transpose it to have each
# mail as a column.
data_matrix = Matrix(clean_words_df)'
```

In the first line, we create a variable *all_words* to store a list of all of the words present in the emails. As our dataset has a column for each word, we do this by storing the names of every column with the *names* function, except for the first and last column, which are for email id and label, respectively.

Let's move on to the second and third lines of the code. We would like to filter out some words that are very common in the English language, such as articles and pronouns, which will most likely add noise rather than information to our classification algorithm. For this we will use two Julia packages that are specially designed for working with texts of any type. These are Languages.jl and TextAnalysis.jl. In the third line, we create a StringDocument, which is a struct provided by TextAnalysis.jl and we use its built-in methods to remove articles and pronouns from the list of words we created before. This is done calling the *prepare* function two times, with two different flags: *strip_articles* and *strip_pronouns*. What follows is just code to recover our original DataFrame with only the relevant columns, i.e., the words that were not filtered. A *clean_words_df* DataFrame is created selecting those columns only. Finally, we turn our DataFrame into a matrix with its rows and columns transposed. This is just the convention used by the packages we are working with to make our analysis; each column is one data realization.

Next, we need to divide the data in two: a training set and a testing set. This is standard practice when working with models that learn from data, like the one we're going to implement. We train the model with the training set, then evaluate the model's accuracy by having it make predictions on the testing set. In Julia, the package MLDataUtils.jl has some nice functionalities for data manipulations like this.

```
labels = raw_df.Prediction
(x_train, y_train), (x_test, y_test) = splitobs(shuffleobs((data_matrix, labels)),
at = 0.7)
```

The function `splitobs` splits our dataset into a training set and a testing set, and `shuffleobs` randomizes the order of the data in the split. We pass a `labels` array

to our split function so it knows how to properly split the dataset. Now we can turn our attention to building the spam filter.

4.4 The Naive Bayes Approach

As we mentioned, what we are facing here is a *classification* problem, and we will code from scratch and use a *supervised learning* algorithm to find a solution with the help of Bayes' theorem. We're going to use a *naive Bayes* classifier to create our spam filter. We're going to use a classifier to create our spam filter. This method is going to treat each email just as a collection of words, with no regard for the order in which they appear. This means we won't take into account semantic considerations like the particular relationship between words and their context.

Our strategy will be to estimate a probability of an incoming email being ham or spam and make a decision based on that. Our general approach can be summarized as:

$$P(spam|email) \propto P(email|spam)P(spam)$$
$$P(ham|email) \propto P(email|ham)P(ham)$$

Notice we use the $\propto$ sign, meaning *proportional to*, instead of the = sign because the denominator from Bayes's theorem is missing. In this case, we won't need to calculate it, as it's the same for both probabilities and all we're going to care about is a comparison of these two probabilities.
In this naive approach, where semantics aren't taken into account and each email is just a collection of words, the conditional probability $P(email|spam)$ means the probability that a given email can be generated with the collection of words that appear in the spam category of our data. Let's take a quick example. Imagine for a moment that our training set of emails consists just of these three emails, all labeled as spam:

- Email 1: 'Are you interested in buying my product?'
- Email 2: 'Congratulations! You've won $1000!'
- Email 3: 'Check out this product!'

Also imagine we receive a new, unclassified email and we want to discover $P(email|spam)$. The new email looks like this:

- New email: 'Apply and win all these products!'

The new email contains the words *win* and *product*, which are rather common in our example's training data. We would therefore expect $P(email|spam)$, the probability of the new email being generated by the words encountered in the training spam email set, to be relatively high.

(The word \emph{win} appears in the form \emph{won} in the training set, but that's OK. The standard linguistic technique of \emph{lemmatization} groups together any related forms of a word and treats them as the same word.)

Mathematically, the way to calculate **P(email|spam)** is to take each word in our target email, calculate the probability of it appearing in spam emails based on our training set, and multiply those probabilties together.

$$\mathbf{P(email|spam)=\prod_{i=1}^{n} P(word_i|spam)}$$

We use a similar calculation to determine **P(email|ham)** the probability of the new email being generated by the words encountered in the training ham email set:

$$\mathbf{P(email|ham)=\prod_{i=1}^{n} P(word_i|ham)}$$

The multiplication of each of the probabilities associated with a particular word here stems from the naive assumption that all the words in the email are statistically independent. In reality, this assumption isn't necessarily true. In fact, it's most likely false. Words in a language are never independent from one another, but this simple assumption seems to be enough for the level of complexity our problem requires.

The probability of a given word $word_i$ being in a given category is calculated like so:

$$P(word_i|spam) = \frac{N_{word_i|spam} + \alpha}{N_{spam} + \alpha N_{vocabulary}}$$

$$P(word_i|ham) = \frac{N_{word_i|ham} + \alpha}{N_{ham} + \alpha N_{vocabulary}}$$

These formulas tell us exactly what we have to calculate from our data. We need the numbers $\mathbf{N_{word_i|spam}}$ and $\mathbf{N_{word_i|ham}}$ for each word, meaning the number of times that $word_i$ is used in the spam and ham categories, respectively. $\mathbf{N_{spam}}$ and $\mathbf{N_{ham}}$ are the total number of words used in the spam and ham categories (including all word repetitions), and $\mathbf{N_{vocabulary}}$ is the total number of unique words in the dataset. The variable $\boldsymbol{\alpha}$ is a smoothing parameter that prevents the probability of a given word being in a given category from going down to zero. If a given word hasn't appeared in the spam category in our training dataset, for example, we don't want to assign it zero probability of appearing in new spam emails.

As all of this information will be specific to our dataset, a clever way to aggregate it is to use a Julia *struct*, with attributes for the pieces of data we'll need to access over and over during the prediction process. Here's the implementation:

```
mutable struct BayesSpamFilter
    words_count_ham::Dict{String, Int64}
    words_count_spam::Dict{String, Int64}
    N_ham::Int64
    N_spam::Int64
```

```
    vocabulary::Array{String}
    BayesSpamFilter() = new()
end
```

The relevant attributes of the struct are `words_count_ham` and `words_count_spam`, two dictionaries containing the frequency of appearance of each word in the ham and spam datasets; `N_ham` and `N_spam`, the total number of words appearing in each category; and `vocabulary`, an array of all the unique words in the dataset.
The line `BayesSpamFilter() = new()` is the constructor of this struct. Because the constructor is empty, all the attributes will be undefined when we instantiate the filter. We'll have to define some functions to fill these variables with values that are relevant to our particular problem. First, here's a function `word_count` that counts the occurrences of each word in the ham and spam categories.
Now we are going to define some functions that will be important for our filter implementation.

```
function words_count(word_data, vocabulary, labels, spam=0)
    count_dict = Dict{String,Int64}()
    n_emails = size(word_data)[2]
    for (i, word) in enumerate(vocabulary)
        count_dict[word] = sum([word_data[i, j] for j in 1:n_emails if labels[j]
== spam])
    end
    return count_dict
end
```

The function *word_count* counts the occurrences of each word in the ham and spam categories. One of its parameters is *word_data*, which we defined before and is a matrix where each column is an email and each row is a word.

Next, we'll define a `fit!` function for our spam filter struct. Notice we're using the bang (`!`) convention here to indicate a function that modifies its arguments in-place (in this case, the spam filter struct itself). This function *fits* our model to the data, a typical procedure in data science and machine learning areas.

```
function fit!(model::BayesSpamFilter, x_train, y_train, voc)
    model.vocabulary = voc
    model.words_count_ham = words_count(x_train, model.vocabulary, y_train, 0)
    model.words_count_spam = words_count(x_train, model.vocabulary, y_train, 1)
    model.N_ham = sum(values(model.words_count_ham))
    model.N_spam = sum(values(model.words_count_spam))
    return
end
## fit! (generic function with 1 method)
```

What we mean by fitting the model to the data is mainly filling all the undefined parameters in our struct with values represented by the training data. To this end, the `words_count` function we defined earlier is used. Notice that we're only fitting the model to the training portion of the data, since we're reserving the testing portion to evaluate the model's accuracy.

4.5 Training the Model

Now it's time to instantiate our spam filter and fit the model to the training data. With the struct and helper functions we've defined, the process is quite straightforward.

```
spam_filter = BayesSpamFilter()
fit!(spam_filter, x_train, y_train, vocabulary)
```

We create an instance of our `BayesSpamFilter` struct and pass it to our `fit!` function along with the data. Notice that we're only passing in the training portion of the dataset, since we want to reserve the testing portion to evaluate the model's accuracy later.

4.6 Making Predictions

Now that we have our model, we can use it to make some spam vs. ham predictions and assess its performance. We'll define a few more functions to help with this process. First, we need a function implementing the TAL formula that we discussed earlier.

```
function word_spam_probability(word, words_count_ham, words_count_spam, N_ham,
N_spam, n_vocabulary, α)
    ham_prob = (words_count_ham[word] + α) / (N_ham + α * (n_vocabulary))
    spam_prob = (words_count_spam[word] + α) / (N_spam + α * (n_vocabulary))
    return ham_prob, spam_prob
end
## word_spam_probability (generic function with 1 method)
```

This function calculates $\mathbf{P(word_i|spam)}$ and $\mathbf{P(word_i|ham)}$ for a given word. We'll call it for each word of an incoming email within another function, `spam_predict`, to calculate the probability of that email being spam or ham.

```
function spam_predict(email, model::BayesSpamFilter, α, tol=100)
    ngrams_email = ngrams(StringDocument(email))
    email_words = keys(ngrams_email)
    n_vocabulary = length(model.vocabulary)
    ham_prior = model.N_ham / (model.N_ham + model.N_spam)
    spam_prior = model.N_spam / (model.N_ham + model.N_spam)

    if length(email_words) > tol
        word_freq = values(ngrams_email)
        sort_idx = sortperm(collect(word_freq), rev=true)
        email_words = collect(email_words)[sort_idx][1:tol]
    end

    email_ham_probability = BigFloat(1)
    email_spam_probability = BigFloat(1)

    for word in intersect(email_words, model.vocabulary)
        word_ham_prob, word_spam_prob = word_spam_probability(word,
model.words_count_ham, model.words_count_spam, model.N_ham, model.N_spam,
n_vocabulary, α)
```

```
        email_ham_probability *= word_ham_prob
        email_spam_probability *= word_spam_prob
    end
    return ham_prior * email_ham_probability, spam_prior * email_spam_probability
end
```

This function takes as input a new email that we want to classify as spam or ham, our fitted model, an αα value (which we've already discussed), and a tolerance value `tol`. The latter sets the maximum number of unique words in an email that we'll look at.

We saw that the calculations for **P(email|spam)** and **P(email|ham)** require the multiplication of each **P(word$_i$|spam)** and **P(word$_i$|ham)** term. When emails consist of a large number of words, this multiplication may lead to very small probabilities, up to the point that the computer interprets those probabilities as zero. This isn't desirable; we need values of **P(email|spam)** and **P(email|ham)** that are larger than zero in order to multiply them by **P(spam)** and **P(ham)** respectively, and compare these values to make a prediction. To avoid probabilities of zero, we'll only consider up to the `tol` most frequently used words in the email.

Finally, we arrive to the point of actually testing our model. We create another function to manage the process. This function classifies each email into Ham (represented by the number 0) or Spam (represented by the number 1)

```
function get_predictions(x_test, y_test, model::BayesSpamFilter, α, tol=200)
    N = length(y_test)
    predictions = Array{Int64,1}(undef, N)
    for i in 1:N
        email = string([repeat(string(word, " "), N) for (word, N) in
zip(model.vocabulary, x_test[:, i])]...)
        pham, pspam = spam_predict(email, model, α, tol)
        pred = argmax([pham, pspam]) - 1
        predictions[i] = pred
    end

    predictions
end
```

This function takes in the testing portion of the data and our trained model. We call our `spam_predict` function for each email in the testing data and use the maximum (`argmax`) of the two returned probability values to predict (`pred`) if the email is spam or ham. We return the predictions as an array of values, which will contain zeros for ham emails, and ones for spam emails. Here we call the function to make predictions about the test data:

```
predictions = get_predictions(x_test, y_test, spam_filter, 1)
```

Let's take a look at the predicted classifications of just the first five emails in the test data.

```
predictions[1:5]
## 5-element Vector{Int64}:
##  0
##  1
##  0
##  0
##  0
```

Of the first five emails, one (the third) was classified as spam, and the rest were classified as ham.

4.7 Evaluating the Accuracy

Looking at the predictions themselves is pretty meaningless; what we really want to know is the model's accuracy. We'll define another function to calculate this.

```
function spam_filter_accuracy(predictions, actual)
    N = length(predictions)
    correct = sum(predictions .== actual)
    accuracy = correct / N
    accuracy
end
```

This function compares the predicted classifications with the actual classifications of the test data, counts the number of correct predictions, and divides this number by the total number of test emails, giving us an accuracy measurement. Here we call the function:

```
spam_filter_accuracy(predictions, y_test)
## 0.9503865979381443
```

The output indicates our model is about 95 percent accurate. It appears our model is performing very well! Such a high accuracy rate is quite astonishing for a model so naive and simple. In fact, it may be a little too good to be true, because we have to take into account one more thing. Our model classifies emails into spam or ham, but the amount of ham emails in our data set is considerably higher than the spam ones. Let's see the percentages:

```
sum(raw_df[!, :Prediction])/length(raw_df[!, :Prediction])
## 0.2900232018561485
```

This tells us that only 30 percent of the emails in the training set are spam: we first sum over the `Prediction` column of the dataset remembering it only consists of 0s and 1s, and then we divide by the total amount of mails. This type of classification problem, where there's an unequal distribution of classes in the dataset, is called *imbalanced*. With unbalanced data, a better way to see how the model is performing is to construct a *confusion matrix*, an $\mathbf{N}\times\mathbf{N}$ matrix, where $\mathbf{N}$ is the number of target classes (in our case, 2, for spam and ham). The matrix compares the actual values for each class with those predicted by the model. Here's a function that builds a confusion matrix for our spam filter:

```
function spam_filter_confusion_matrix(y_test, predictions)
    # 2x2 matrix is instantiated with zeros
    confusion_matrix = zeros((2, 2))

    confusion_matrix[1, 1] = sum(isequal(y_test[i], 0) & isequal(predictions[i],
0) for i in 1:length(y_test))
    confusion_matrix[1, 2] = sum(isequal(y_test[i], 1) & isequal(predictions[i],
0) for i in 1:length(y_test))
    confusion_matrix[2, 1] = sum(isequal(y_test[i], 0) & isequal(predictions[i],
1) for i in 1:length(y_test))
```

```
    confusion_matrix[2, 2] = sum(isequal(y_test[i], 1) & isequal(predictions[i],
1) for i in 1:length(y_test))

    # Now we convert the confusion matrix into a DataFrame
    confusion_df = DataFrame(prediction=String[], ham_mail=Int64[],
spam_mail=Int64[])
    confusion_df = vcat(confusion_df, DataFrame(prediction="Model predicted Ham",
ham_mail=confusion_matrix[1, 1], spam_mail=confusion_matrix[1, 2]))
    confusion_df = vcat(confusion_df, DataFrame(prediction="Model predicted Spam",
ham_mail=confusion_matrix[2, 1], spam_mail=confusion_matrix[2, 2]))

    return confusion_df
end
```

Now let's call our function to build the confusion matrix for our model.

```
confusion_matrix = spam_filter_confusion_matrix(y_test[:], predictions)
## 2×3 DataFrame
##  Row │ prediction            ham_mail  spam_mail
##      │ String                Float64   Float64
## ─────┼──────────────────────────────────────────
##    1 │ Model predicted Ham     1078.0       37.0
##    2 │ Model predicted Spam      40.0      397.0
```

Row 1 of the confusion matrix shows us all the times our model classified emails to be ham; 1,056 of those classifications were correct and 36 were incorrect. Similarly, the `spam_mail` column shows us the classifications for all the spam emails; 36 were misidentified as ham, and 427 were correctly identified as spam.

Now that we have the confusion matrix, we can calculate the accuracy of the model segmented by category.

```
ham_accuracy = confusion_matrix[1, :ham_mail] / (confusion_matrix[1, :ham_mail] +
confusion_matrix[2, :ham_mail])
## 0.964221824686941
spam_accuracy = confusion_matrix[2, :spam_mail] / (confusion_matrix[1, :spam_mail]
+ confusion_matrix[2, :spam_mail])
## 0.9147465437788018
```

With this values now we have more fine-grained measure of the accuracy of our model. Now we know that it is not the same for our filter to make a prediction over a spam or a ham email. As a consequence of the inbalance in our data, ham emails will be classified as such more accurately than spam emails. Still, with both percentages above 90, are some pretty good values for a model so simple and naive. Models like these can be used like a baseline for creating more complex ones on top of them.

4.8 Summary

In this chapter, we've used a naive Bayes approach to build a simple email spam filter. We walked through the whole process of training, testing, and evaluating a learning model. First, we obtained a dataset of emails already classified as spam or ham and preprocessed the data. Then we considered the theoretical framework for our naive analysis. Using Bayes's theorem on the data available, we assigned a probability of belonging to a spam or ham email to each word of the email dataset.

The probability of a new email being classified as spam is therefore the product of the probabilities of each of its constituent words. We defined a Julia `struct` for the spam filter object and created functions to fit the spam filter object to the data. Finally, we made predictions on new data and evaluated our model's performance by calculating the accuracy and making a confusion matrix.

Why won't linear regression work for spam filtering?

Motivation

Let's say you're given **10,000** emails, each labeled as **spam** or **not spam**, and the wide range of vocabulary of the emails spans **100,000** words total. Should you use linear regression on the labeled emails to build a spam filter? **No**. Let's learn why.

Turning this into a math problem

First, let's set this up as a math problem.

- **m** = number of training samples
- **n** = number of features

In this problem, $m=10,000$ and $n=100,000$. Let's represent a single email by a **feature vector**:

$$x=[100:1:0]\text{aabilityable:bargain:zap}=[100:1:0]\text{aabilityable:bargain:zap}$$

A **1** in the i-th position of the feature vector means the email has the corresponding i-th word (i.e., a **1** in the **1st** position means the word **"a"** exists in the email). On the other hand, a **0** means the word doesn't appear. In the example above, the feature vector shows the email contains the words **"a"** and **"bargain"** but not **"ability", "able",** and **"zap"**. Notice that the feature vector doesn't take into account the *order* with which the words appear in an email.

If you were to apply linear regression on this problem, the goal would be to find 100,000 **weights**. Each of these weights $(\theta_1,\theta_2,\cdots,\theta_{100,000})$ controls how much influence each of the words in the email have on the decision to classify that email as spam or not. For example, if $\theta_2=0$, then it's telling us the **2nd** word in the feature vector, the word **"ability"**, doesn't give us more information about whether or not an email is spam.

Mathematically, we want the dot product of the **weight vector** and the feature vector, or $x^T\theta$. x^T is a $1\times n$ vector and θ is a $n\times 1$ vector, so their dot product is a 1×1 vector, which is a scalar called the **hypothesis**, and it's often denoted by $h_\theta(x)$. The hypothesis can be any of a range of values. It can be **20, 102.5, 900.1092,** or **1,000,000**.

Reason 1: The hypothesis's range should be {0, 1}

This brings us to the first folly of using linear regression to build a spam filter. The hypothesis should be a value 1 or 0 (**spam** or **not spam**, respectively). Yet linear regression allows it be any real number. Mathematically, we want $h_\theta(x)\in\{0,1\}$, yet with linear regression we get $h_\theta(x)\in\mathbb{R}$.

Reason 2: What is a good threshold value b?

Because $h_\theta(x)\in\mathbb{R}$, we have to define a function that allows us to transform $h_\theta(x)$ into **0** or **1**.

$$f(h_\theta(x))=\begin{cases}1 & \text{if } h_\theta(x)>b\\ 0 & \text{if } h_\theta(x)\leq b\end{cases}$$

where **b** is a threshold value. If the hypothesis is above **b**, we'll say the email is not spam. If it's below or equal to **b**, we'll say it is spam. Now comes the question, what should **b** be? And even if you determine **b**, if you add a new email to your set of emails with which you built a spam filter, the new email could dramatically change the **weight vector** and alter how your spam filter perceives existing email. Watch this excellent explanation by Andrew Ng if you're still confused.

Reason 3: The normal equations cannot be solved

If we stack all the emails, we get an $\mathbf{m\times n}$ matrix

$$X=\begin{bmatrix} -- & (x^{(1)})^T & -- \\ -- & (x^{(2)})^T & -- \\ & \vdots & \\ -- & (x^{(m)})^T & -- \end{bmatrix}$$

The label for each email

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

To find the optimal weight vector θ, use the following equation

$$\theta=(X^TX)^{-1}X^Ty$$

This is called the **normal equation**. Unfortunately, for our problem, it can't be solved because X^TXhas no inverse.

Why Won't Linear Regression Work for Filtering Spam?

Because we're already familiar with linear regression and that's a tool in our toolbelt, let's start by talking through what we'd need to do in order to try to use linear regression. We already know that's *not* what we're going to use, but let's talk it through to get to why. Imagine a dataset or matrix where each row represents a different email message (it could be keyed by email_id). Now let's make each word in the email a *feature*—this means that we create a column called "Viagra," for example, and then for any message that has the word Viagra in it at least once, we put a 1 in; otherwise we assign a 0. Alternatively, we could put the number of times the word appears. Then each column represents the appearance of a different word.

Thinking back to the previous chapter, in order to use linear regression, we need to have a training set of emails where the messages have *already* been labeled with some outcome variable. In this case, the outcomes are either spam or not. We could do this by having human evaluators label messages "spam," which is a reasonable, albeit time-intensive, solution. Another way

to do it would be to take an existing spam filter such as Gmail's spam filter and use those labels. (Now of course if you already had a Gmail spam filter, it's hard to understand why you might also want to build another spam filter in the first place, but let's just say you do.) Once you build a model, email messages would come in without a label, and you'd use your model to predict the labels.

The first thing to consider is that your target is binary (0 if not spam, 1 if spam)—you wouldn't get a 0 or a 1 using linear regression; you'd get a number. Strictly speaking, this option really isn't ideal; linear regression is aimed at modeling a continuous output and this is binary.

This issue is basically a nonstarter. We should use a model appropriate for the data. But if we wanted to fit it in R, in theory it could still work. R doesn't check for us whether the model is appropriate or not. We could go for it, fit a linear model, and then use that to predict and then choose a critical value so that above that predicted value we call it "1" and below we call it "0."

But if we went ahead and tried, it still wouldn't work because there are too many variables compared to observations! We have on the order of 10,000 emails with on the order of 100,000 words. This won't work. Technically, this corresponds to the fact that the matrix in the equation for linear regression is not invertible—in fact, it's not even close. Moreover, maybe we can't even store it because it's so huge.

Maybe we could limit it to the top 10,000 words? Then we could at least have an invertible matrix. Even so, that's too many variables versus observations to feel good about it. With carefully chosen feature selection and domain expertise, we could limit it to 100 words and that could be enough! But again, we'd still have the issue that linear regression is not the appropriate model for a binary outcome.

How About k-nearest Neighbors?

We're going to get to Naive Bayes shortly, we promise, but let's take a minute to think about trying to use k-nearest neighbors (k-NN) to create a spam filter. We would still need to choose features, probably corresponding to words, and we'd likely define the value of those features to be 0 or 1, depending on whether the word is present or not. Then, we'd need to define when two emails are "near" each other based on which words they both contain.

Again, with 10,000 emails and 100,000 words, we'll encounter a problem, different from the noninvertible matrix problem. Namely, the space we'd be working in has *too many dimensions*. Yes, computing distances in a 100,000-dimensional space requires lots of computational work. But that's not the real problem.

The real problem is even more basic: even our nearest neighbors are really far away. This is called "the curse of dimensionality," and it makes k-NN a poor algorithm in this case.

1.9. Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable **y** and dependent feature vector $\mathbf{x_1}$ through $\mathbf{x_{n1}}$:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Using the naive conditional independence assumption that

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y),$$

for all **i**, this relationship is simplified to

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

Since $P(x_1, \ldots, x_n)$ is constant given the input, we can use the following classification rule:

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

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

and we can use Maximum A Posteriori (MAP) estimation to estimate **p(y)** and the $P(x_i \mid y)$ former is then the relative frequency of class **y** in the training set.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters. (For theoretical reasons why naive Bayes works well, and on which types of data it does, see the references below.)

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator, so the probability outputs from `predict_proba` are not to be taken too seriously.

Naive Bayes is a probabilistic algorithm based on the Bayes Theorem used for email spam filtering in data analytics. If you have an email account, we are sure that you have seen emails being categorised into different buckets and automatically being marked important, spam, promotions, etc. Isn't it wonderful to see machines being so smart and doing the work for you?

More often than not, these labels added by the system are right. So does this mean our email software is reading through every communication and now understands what you as a user would have done? Absolutely right! In this age and time of data analytics & machine learning, automated filtering of emails happens via algorithms like Naive Bayes Classifier, which apply the basic Bayes Theorem on the data.

In this article, we will understand briefly about the Naive Bayes Algorithm before we get our hands dirty and analyse a real email dataset in Python. This blog is second in the series to understand the Naive Bayes Algorithm.

The Naive Bayes Classifier Formula

One of the most simple yet powerful classifier algorithms, Naive Bayes is based on Bayes' Theorem Formula with an assumption of independence among predictors. Given a Hypothesis A and evidence B, Bayes' Theorem calculator states that the relationship between the probability of Hypothesis before getting the evidence P(A) and the probability of the hypothesis after getting the evidence P(A|B) is:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Source: Wikipedia

Here:

- A, B = events
- P(A|B) = probability of A given B is true
- P(B|A) = probability of B given A is true
- P(A), P(B) = the independent probabilities of A and B

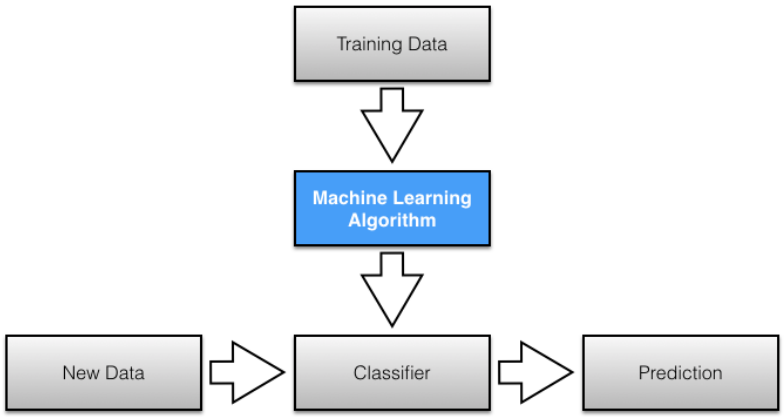
This theorem, as explained in one of our previous articles, is mainly used for classification techniques in data analytics. The Naive Bayes theorem calculator pays an important role in spam detection of emails.

Detecting Email Spam

Modern spam filtering software continuously struggles to categorise the emails correctly. Unwanted spam & promotional communication is the toughest of them all. Spam communication algorithms must be iterated continuously since there is an ongoing battle between spam filtering software and anonymous spam & promotional mail senders. Naive Bayes Algorithm in data analytics forms the base for text filtering in Gmail, Yahoo Mail, Hotmail & all other platforms.

Like Naive Bayes, other classifier algorithms like Support Vector Machine, or Neural Network also get the job done! Before we begin, here is the dataset for you to download:

Email Spam Filtering Using Naive Bayes Algorithm
This would be a zipped file, attached in the email. Please allow users to download this data.



Probability is all around us. It's in the weather updates we receive daily, to sports team strategies we see on TV, to insurance premiums we purchase. It is also heavily prevalent in the field of Machine Learning, as one of the mathematical driving forces powering modern algorithms.

It also powers earlier algorithms, such as the Naïve Bayes Classifier. This was a unique approach to applying Bayes' Theorem to real world problems when the classifier was first devised in the 1990s, and is best known for it's application as an early method of spam filtering

Bayes' theorem was invented by Thomas Bayes in 1763, when he published a work titled *An Essay towards solving a Problem in the Doctrine of Chances* (1763). In this essay, Bayes describes how conditional probability can be used to estimate the likelihood of certain events occurring, given certain external events have also occurred. The formula for the theorem is as follows:

Where:

- $Pr(A)$ = the probability of event A occurring,
- $Pr(B)$ = the probability of event B occurring,
- $Pr(A|B)$ = Conditional Probability - the probability of event A occurring, given event B also occurs,

- *Pr(B|A)* = Conditional Probability - the probability of event *B* occurring, given event *A* also occurs.

This formula can alternatively be expanded out into:

This alternative equation also computes *Pr(A|B)*, but is more useful in the context of spam filtering than the original simpler equation

The concept of spam filtering is simple - detect spam emails from authentic (non-spam/ham) emails. To do this, the goal would be to get a measure of how 'spammy' an incoming email is. The extended form of Bayes' Rule comes into play here.

With Bayes' Rule, we want to find the probability an email is spam, given it contains certain words. We do this by finding the probability that **each word in the email is spam**, and then **multiply these probabilities together** to get the overall email spam metric to be used in classification.

The the probability of an email being spam ***S*** given a certain word ***W*** appears is defined by the left hand side of the above equation ***Pr(S|W)***.

The right hand side gives the formula to compute this probability. This is:

- the probability the word occurs in the email given it is a spam email ***Pr(W|S)*** multiplied by the probability of an email being spam ***Pr(S)***,
- divided the probability the word occurs in the email given it is a spam email multiplied by the probability of an email being spam,
- plus the probability the word occurs in the email given it is a **non-spam** email ***Pr(W|¬S)*** multiplied by the probability of an email being **non-spam** ***Pr(¬S)***.

Probabilities can range between 0 and 1. For this spam filter, we will define that any email with a total 'spaminess' metric of over 0.5 (50%) will be deemed a spam email.

When the ***Pr(S|W)*** has been found for each word in the email, they are multiplied together to give the overall probability that the email is spam. If this probability is over the 'spam threshold' of **0.5**, the email is classified as a spam email.

Coding a Spam Filter from Scratch:

ML libraries such as scikit-learn are brilliant for testing out-of-the box algorithms on your data. However it can be beneficial to explore the inner workings of an algorithm in order to better understand and appreciate it by writing it from scratch. Here we will write an implementation of Naïve Bayes Classifier for Spam Filtering in pure python, without the aid of external libraries.

Training and testing data:

Define some training and test data for each class, spam and ham. These emails will be short for brevity and clarity.

```
train_spam = ['send us your password', 'review our website', 'send your password', 'send us your account']
train_ham = ['Your activity report','benefits physical activity', 'the importance vows']
test_emails = {'spam':['renew your password', 'renew your vows'], 'ham':['benefits of our account', 'the importance of physical activity']}
```

Iterate over the labelled spam emails and, for each word w in the entire training set, count how many of the spam emails contain w:

```
# make a vocabulary of unique words that occur in known spam emails

vocab_words_spam = []

for sentence in train_spam:
    sentence_as_list = sentence.split()
    for word in sentence_as_list:
        vocab_words_spam.append(word)

print(vocab_words_spam)['send', 'us', 'your', 'password', 'review', 'our', 'website', 'send', 'your', 'password', 'send', 'us', 'your', 'account']
```

Convert each list element to a dictionary key. This will delete duplicates, as dictionaries cannot have multiple keys with the same name. Convert remaining keys back to list:

```
vocab_unique_words_spam = list(dict.fromkeys(vocab_words_spam))
print(vocab_unique_words_spam)['send', 'us', 'your', 'password', 'review', 'our', 'website', 'account']
```

How do you determine how 'spammy' a word is?

According to this overview on Naïve Bayes spam filtering, 'Spamicity' can be calculated by taking the total number of emails that have already been hand-labelled as either spam or ham, and using that data to compute word spam probabilities, by counting the **frequency of each word**.

This uses the same type of conditional probability found in Bayes' Rule. We can count how many spam emails have the word “send” and divide that by the total number of spam emails - this gives a measure of the word's 'spamicity', or how likely it is to be in a spam email.

The same approach can be applied for evaluating how 'hamicity' or authenticity of a word.

Smoothing

Smoothing should be applied to avoid the scenario that a word would appear **zero times** in the spam training examples, but would appear in the ham training example, or vice-versa.

We want to avoid this scenario, as this would lead to a 0 as the numerator when performing the spamicity calculation: zero divided by anything is zero. A spam-bot could send us an email with this word, and the final email spam calculation would be returned as **0**, when we multiplied all the other word spam probabilities together - any other

numbers multiplied by zero will be zero. This fraudulent email now having 0% spamicity would be classified as ham, and pass quietly into our inbox.

The solution is to **add 1 to every word count**, so there will never be a zero word count. **Add 2 to the denominator** to account for this increase to the numerator. We add 2 here in order to account for the number of classes we have - a spam class and a ham class. This would keep the total probability at 1.0 if you were to add up all the individual word probabilities together.

In this example, the word "send" has a spamicity of 75% (**66% with smoothing applied**.) This is over the defined spam threshold of 0.5, meaning it has a high probability of being 'spammy' word.

```
dict_spamicity = {}
for w in vocab_unique_words_spam:
    emails_with_w = 0     # counter
    for sentence in train_spam:
        if w in sentence:
            emails_with_w+=1

    print(f"Number of spam emails with the word {w}:
{emails_with_w}")
    total_spam = len(train_spam)
    spamicity = (emails_with_w+1)/(total_spam+2)
    print(f"Spamicity of the word '{w}': {spamicity} \n")
    dict_spamicity[w.lower()] = spamicityNumber of spam emails with
the word send: 3
Spamicity of the word 'send': 0.6666666666666666

Number of spam emails with the word us: 2
Spamicity of the word 'us': 0.5

Number of spam emails with the word your: 3
Spamicity of the word 'your': 0.6666666666666666
```

```
Number of spam emails with the word password: 2
Spamicity of the word 'password': 0.5

Number of spam emails with the word review: 1
Spamicity of the word 'review': 0.3333333333333333

Number of spam emails with the word our: 4
Spamicity of the word 'our': 0.8333333333333334

Number of spam emails with the word website: 1
Spamicity of the word 'website': 0.3333333333333333

Number of spam emails with the word account: 1
Spamicity of the word 'account': 0.3333333333333333
print(dict_spamicity){'send': 0.6666666666666666, 'us': 0.5,
'your': 0.6666666666666666, 'password': 0.5, 'review':
0.3333333333333333, 'our': 0.8333333333333334, 'website':
0.3333333333333333, 'account': 0.3333333333333333}
```

Calculate Hamicity of non-spam words:

```
# make a vocabulary of unique words that occur in known ham emails

vocab_words_ham = []

for sentence in train_ham:
    sentence_as_list = sentence.split()
    for word in sentence_as_list:
        vocab_words_ham.append(word)

vocab_unique_words_ham = list(dict.fromkeys(vocab_words_ham))
print(vocab_unique_words_ham)['Your', 'activity', 'report',
'benefits', 'physical', 'the', 'importance', 'vows']dict_hamicity =
{}
for w in vocab_unique_words_ham:
    emails_with_w = 0     # counter
    for sentence in train_ham:
        if w in sentence:
            print(w+":", sentence)
            emails_with_w+=1

    print(f"Number of ham emails with the word '{w}':
{emails_with_w}")
    total_ham = len(train_ham)
    Hamicity = (emails_with_w+1)/(total_ham+2)       # Smoothing
applied
    print(f"Hamicity of the word '{w}': {Hamicity} ")
```

```
    dict_hamicity[w.lower()] = Hamicity
                                        # Use built-in lower() to
keep all words lower case - useful later when
                                        # comparing spamicity vs
hamicity of a single word - e.g. 'Your' and
                                        # 'your' will be treated
as 2 different words if not normalized to lower
# case.Your: Your activity report
Number of ham emails with the word 'Your': 1
Hamicity of the word 'Your': 0.4
activity: Your activity report
activity: benefits physical activity
Number of ham emails with the word 'activity': 2
Hamicity of the word 'activity': 0.6
report: Your activity report
Number of ham emails with the word 'report': 1
Hamicity of the word 'report': 0.4
benefits: benefits physical activity
Number of ham emails with the word 'benefits': 1
Hamicity of the word 'benefits': 0.4
physical: benefits physical activity
Number of ham emails with the word 'physical': 1
Hamicity of the word 'physical': 0.4
the: the importance vows
Number of ham emails with the word 'the': 1
Hamicity of the word 'the': 0.4
importance: the importance vows
Number of ham emails with the word 'importance': 1
Hamicity of the word 'importance': 0.4
vows: the importance vows
Number of ham emails with the word 'vows': 1
Hamicity of the word 'vows': 0.4print(dict_hamicity){'your': 0.4,
'activity': 0.6, 'report': 0.4, 'benefits': 0.4, 'physical': 0.4,
'the': 0.4, 'importance': 0.4, 'vows': 0.4}
```

Compute Probability of Spam P(S):

This computes the probability of any one email being spam, by dividing the total number of spam emails by the total number of all emails.

```
prob_spam = len(train_spam) / (len(train_spam)+(len(train_ham)))
print(prob_spam)0.5714285714285714
```

Compute Probability of Ham P(¬S):

This computes the probability of any one email being ham, by dividing the total number of ham emails by the total number of all emails.

```
prob_ham = len(train_ham) / (len(train_spam)+(len(train_ham)))
print(prob_ham)0.42857142857142855
```

Given a set of un-labelled test emails, iterate over each, and create list of distinct words:

```
tests = []
for i in test_emails['spam']:
    tests.append(i)

for i in test_emails['ham']:
    tests.append(i)

print(tests)
['renew your password', 'renew your vows', 'benefits of our
account', 'the importance of physical activity']# split emails into
distinct words

distinct_words_as_sentences_test = []

for sentence in tests:
    sentence_as_list = sentence.split()
    senten = []
    for word in sentence_as_list:
        senten.append(word)
    distinct_words_as_sentences_test.append(senten)

print(distinct_words_as_sentences_test)[['renew', 'your',
'password'], ['renew', 'your', 'vows'], ['benefits', 'of', 'our',
'account'], ['the', 'importance', 'of', 'physical',
'activity']]test_spam_tokenized =
[distinct_words_as_sentences_test[0],
distinct_words_as_sentences_test[1]]
test_ham_tokenized = [distinct_words_as_sentences_test[2],
distinct_words_as_sentences_test[3]]
print(test_spam_tokenized)[['renew', 'your', 'password'], ['renew',
'your', 'vows']]
```

Ignore the words that you haven't seen in the labelled training data:

The reason you would ignore totally unseen words is that you have nothing to really base their level of spamicity or hamicity off. Simply dropping them therefore will not affect the overall probability one way or another, and will not have a significant impact on the class the algorithm selects. It makes more sense to drop unseen words than to add a 1-count with smoothing therefore.

```
reduced_sentences_spam_test = []
for sentence in test_spam_tokenized:
    words_ = []
    for word in sentence:
        if word in vocab_unique_words_spam:
            print(f"'{word}', ok")
            words_.append(word)
        elif word in vocab_unique_words_ham:
            print(f"'{word}', ok")
            words_.append(word)
        else:
            print(f"'{word}', word not present in labelled spam
training data")
    reduced_sentences_spam_test.append(words_)
print(reduced_sentences_spam_test)'renew', word not present in
labelled spam training data
'your', ok
'password', ok
'renew', word not present in labelled spam training data
'your', ok
'vows', ok
[['your', 'password'], ['your', 'vows']]reduced_sentences_ham_test
= []                    # repeat for ham words
for sentence in test_ham_tokenized:
    words_ = []
    for word in sentence:
        if word in vocab_unique_words_ham:
            print(f"'{word}', ok")
            words_.append(word)
        elif word in vocab_unique_words_spam:
            print(f"'{word}', ok")
            words_.append(word)
        else:
```

```
            print(f"'{word}', word not present in labelled ham
training data")
    reduced_sentences_ham_test.append(words_)
print(reduced_sentences_ham_test)'benefits', ok
'of', word not present in labelled ham training data
'our', ok
'account', ok
'the', ok
'importance', ok
'of', word not present in labelled ham training data
'physical', ok
'activity', ok
[['benefits', 'our', 'account'], ['the', 'importance', 'physical',
'activity']]
```

Stemming - remove non-key words:

Removal of non-key words can help the classifier focus on what words are most important. Deciding what words to remove in this specific case can involve some trial and error, as including or excluding words from a small toy dataset like this will influence the overall classification result.

```
test_spam_stemmed = []
non_key = ['us',  'the', 'of','your']       # non-key words,
gathered from spam,ham and test sentences
for email in reduced_sentences_spam_test:
    email_stemmed=[]
    for word in email:
        if word in non_key:
            print('remove')
        else:
            email_stemmed.append(word)
    test_spam_stemmed.append(email_stemmed)

print(test_spam_stemmed)
remove
remove
[['password'], ['vows']]test_ham_stemmed = []
non_key = ['us',  'the', 'of', 'your']
for email in reduced_sentences_ham_test:
    email_stemmed=[]
    for word in email:
```

```
        if word in non_key:
            print('remove')
        else:
            email_stemmed.append(word)
    test_ham_stemmed.append(email_stemmed)

print(test_ham_stemmed)
remove
[['benefits', 'our', 'account'], ['importance', 'physical',
'activity']]
```

Bayes' Rule

We now use the formula for Bayes' Rule to compute the probability of spam given a certain word from an email. We have already calculated all the necessary probabilities and conditional probabilities needed for the right-hand side of the equation.

Computing the overall probability for the entire email is then obtained by multiplying of all these individual word probabilities together.

This approach speaks to the reason this Bayes' classifier is **'Naïve'** - it does not take into account the relationship of any one word to the next. It assumes there is no order to the words appearing in a sentence, and that they are all **independent** of each other, as if language was a random pass though a dictionary. This of course is not the case, which is why this early form of spam filtering has been overtaken by more advanced forms of NLP, such as using word embeddings.

Classify spam test emails:

```
def mult(list_) :        # function to multiply all word probs
together
```

```
    total_prob = 1
    for i in list_:
         total_prob = total_prob * i
    return total_prob

def Bayes(email):
    probs = []
    for word in email:
        Pr_S = prob_spam
        print('prob of spam in general ',Pr_S)
        try:
            pr_WS = dict_spamicity[word]
            print(f'prob "{word}"  is a spam word : {pr_WS}')
        except KeyError:
            pr_WS = 1/(total_spam+2)  # Apply smoothing for word
not seen in spam training data, but seen in ham training
            print(f"prob '{word}' is a spam word: {pr_WS}")

        Pr_H = prob_ham
        print('prob of ham in general ', Pr_H)
        try:
            pr_WH = dict_hamicity[word]
            print(f'prob "{word}" is a ham word: ',pr_WH)
        except KeyError:
            pr_WH = (1/(total_ham+2))  # Apply smoothing for word
not seen in ham training data, but seen in spam training
            print(f"WH for {word} is {pr_WH}")
            print(f"prob '{word}' is a ham word: {pr_WH}")

        prob_word_is_spam_BAYES =
(pr_WS*Pr_S)/((pr_WS*Pr_S)+(pr_WH*Pr_H))
        print('')
        print(f"Using Bayes, prob the the word '{word}' is spam:
{prob_word_is_spam_BAYES}")
        print('############################')
        probs.append(prob_word_is_spam_BAYES)
    print(f"All word probabilities for this sentence: {probs}")
    final_classification = mult(probs)
    if final_classification >= 0.5:
        print(f'email is SPAM: with spammy confidence of
{final_classification*100}%')
    else:
        print(f'email is HAM: with spammy confidence of
{final_classification*100}%')
    return final_classificationfor email in test_spam_stemmed:
    print('')
    print(f"           Testing stemmed SPAM email {email} :")
```

```
    print('                  Test word by word: ')
    all_word_probs = Bayes(email)
    print(all_word_probs)
           Testing stemmed SPAM email ['password'] :
                 Test word by word:
prob of spam in general  0.5714285714285714
prob "password"  is a spam word : 0.5
prob of ham in general  0.42857142857142855
WH for password is 0.2
prob 'password' is a ham word: 0.2

Using Bayes, prob the the word 'password' is spam:
0.7692307692307692
############################
All word probabilities for this sentence: [0.7692307692307692]
email is SPAM: with spammy confidence of 76.92307692307692%
0.7692307692307692

           Testing stemmed SPAM email ['vows'] :
                 Test word by word:
prob of spam in general  0.5714285714285714
prob 'vows' is a spam word: 0.16666666666666666
prob of ham in general  0.42857142857142855
prob "vows" is a ham word:  0.4

Using Bayes, prob the the word 'vows' is spam: 0.35714285714285715
############################
All word probabilities for this sentence: [0.35714285714285715]
email is HAM: with spammy confidence of 35.714285714285715%
0.35714285714285715
```

Result: 1 out of 2 SPAM emails correctly classified.

Next we test how likely the stemmed HAM test emails are to be SPAM.

- This should be closer to zero than to 0.5, as this is an example of finding the probability of SPAM for emails that contain only words not previously seen in HAM training data,

- (as the stemmed HAM words have all never had their 'spamicity' calculated for the spamicity dictionary, therefore stemming will add a count of just 1 to these words, giving them a low probability and low spamicity.)

```
for email in test_ham_stemmed:
    print('')
    print(f"           Testing stemmed HAM email {email} :")
    print('                Test word by word: ')
    all_word_probs = Bayes(email)
    print(all_word_probs)Testing stemmed HAM email ['benefits',
'our', 'account'] :
                  Test word by word:
prob of spam in general  0.5714285714285714
prob 'benefits' is a spam word: 0.16666666666666666
prob of ham in general  0.42857142857142855
prob "benefits" is a ham word:  0.4

Using Bayes, prob the the word 'benefits' is spam:
0.35714285714285715
############################
prob of spam in general  0.5714285714285714
prob "our"  is a spam word : 0.8333333333333334
prob of ham in general  0.42857142857142855
WH for our is 0.2
prob 'our' is a ham word: 0.2

Using Bayes, prob the the word 'our' is spam: 0.847457627118644
############################
prob of spam in general  0.5714285714285714
prob "account"  is a spam word : 0.3333333333333333
prob of ham in general  0.42857142857142855
WH for account is 0.2
prob 'account' is a ham word: 0.2

Using Bayes, prob the the word 'account' is spam: 0.689655172413793
############################
All word probabilities for this sentence: [0.35714285714285715,
0.847457627118644, 0.689655172413793]
email is HAM: with spammy confidence of 20.873340569424727%
0.20873340569424728

           Testing stemmed HAM email ['importance', 'physical',
'activity'] :
                  Test word by word:
prob of spam in general  0.5714285714285714
```

```
prob 'importance' is a spam word: 0.16666666666666666
prob of ham in general  0.42857142857142855
prob "importance" is a ham word:  0.4

Using Bayes, prob the the word 'importance' is spam:
0.35714285714285715
############################
prob of spam in general  0.5714285714285714
prob 'physical' is a spam word: 0.16666666666666666
prob of ham in general  0.42857142857142855
prob "physical" is a ham word:  0.4

Using Bayes, prob the the word 'physical' is spam:
0.35714285714285715
############################
prob of spam in general  0.5714285714285714
prob 'activity' is a spam word: 0.16666666666666666
prob of ham in general  0.42857142857142855
prob "activity" is a ham word:  0.6

Using Bayes, prob the the word 'activity' is spam:
0.2702702702702703
############################
All word probabilities for this sentence: [0.35714285714285715,
0.35714285714285715, 0.2702702702702703]
email is HAM: with spammy confidence of 3.447324875896305%
0.03447324875896305
```

Conclusions:

Naïve Bayes is a useful algorithm in many respects, especially for solving low-data text classification problems. The way in which it can make accurate predictions with limited training data by assuming naïve independence of words is its main advantage, but ironically also a disadvantage when comparing it to more advanced forms of NLP. Other approaches, such as classifiers which use word embeddings, can encode meaning such as context from sentences to obtain more accurate predictions. Still, there is no denying that Naïve Bayes can act as a powerful spam filter.

What Is Data Wrangling?

Data wrangling is the process of removing errors and combining complex data sets to make them more accessible and easier to analyze. Due to the rapid expansion of the amount of data and data sources available today, storing and organizing large quantities of data for analysis is becoming increasingly necessary.

A data wrangling process, also known as a data munging process, consists of reorganizing, transforming and mapping data from one "raw" form into another in order to make it more usable and valuable for a variety of downstream uses including analytics.

Data wrangling can be defined as the process of cleaning, organizing, and transforming raw data into the desired format for analysts to use for prompt decision-making. Also known as data cleaning or data munging, data wrangling enables businesses to tackle more complex data in less time, produce more accurate results, and make better decisions. The exact methods vary from project to project depending upon your data and the goal you are trying to achieve. More and more organizations are increasingly relying on data wrangling tools to make data ready for downstream analytics

Importance of Data Wrangling

Did you know, data professionals spend almost 80% of their time wrangling the data, leaving a mere 20% for exploration and modeling?

Some may question if the amount of work and time devoted to data wrangling is worth the effort. A simple analogy will help you understand. The foundation of a skyscraper is expensive and time-consuming before the above-ground structure starts. Still, this solid foundation is extremely valuable for the building to stand tall and serve its purpose for decades. Similarly, for data handling, once the code and infrastructure foundation are gathered, it will deliver immediate results (sometimes almost instantly) for as long as the process is relevant. However, skipping necessary data wrangling steps will lead to significant downfalls, missed opportunities, and erroneous models that damage the reputation of analysis within the organization.

Benefits of Data Wrangling

- Data wrangling helps to improve data usability as it converts data into a compatible format for the end system.
- It helps to quickly build data flows within an intuitive user interface and easily schedule and automate the data-flow process.
- Integrates various types of information and their sources (like databases, web services, files, etc.)
- Help users to process very large volumes of data easily and easily share data-flow techniques.

Data Wrangling Tools

There are different tools for data wrangling that can be used for gathering, importing, structuring, and cleaning data before it can be fed into analytics and BI apps. You can use automated tools for data wrangling, where the software allows you to validate data mappings and scrutinize data samples at every step of the transformation process. This helps to quickly detect and correct errors in data mapping. Automated data cleaning becomes necessary in businesses dealing with exceptionally large data sets. For manual data cleaning processes, the data team or data scientist is responsible for wrangling. In smaller setups, however, non-data professionals are responsible for cleaning data before leveraging it.

Some examples of basic data munging tools are:

- Spreadsheets / Excel Power Query - It is the most basic manual data wrangling tool
- OpenRefine - An automated data cleaning tool that requires programming skills
- Tabula – It is a tool suited for all data types
- Google DataPrep – It is a data service that explores, cleans, and prepares data
- Data wrangler – It is a data cleaning and transforming tool

Data Wrangling Examples

Data wrangling techniques are used for various use-cases. The most commonly used examples of data wrangling are for:

- Merging several data sources into one data-set for analysis
- Identifying gaps or empty cells in data and either filling or removing them
- Deleting irrelevant or unnecessary data
- Identifying severe outliers in data and either explaining the inconsistencies or deleting them to facilitate analysis

Businesses also use data wrangling tools to

- Detect corporate fraud
- Support data security
- Ensure accurate and recurring data modeling results
- Ensure business compliance with industry standards
- Perform Customer Behavior Analysis
- Reduce time spent on preparing data for analysis
- Promptly recognize the business value of your data
- Find out data trends

- **What is Web Scraping?**
- Web scraping is a technique that is used to extract data from websites. It is also called as Web harvesting.

- This extracted data is saved either in a local file to the computer or to the database. It is the process in which data is collected automatically for the web.

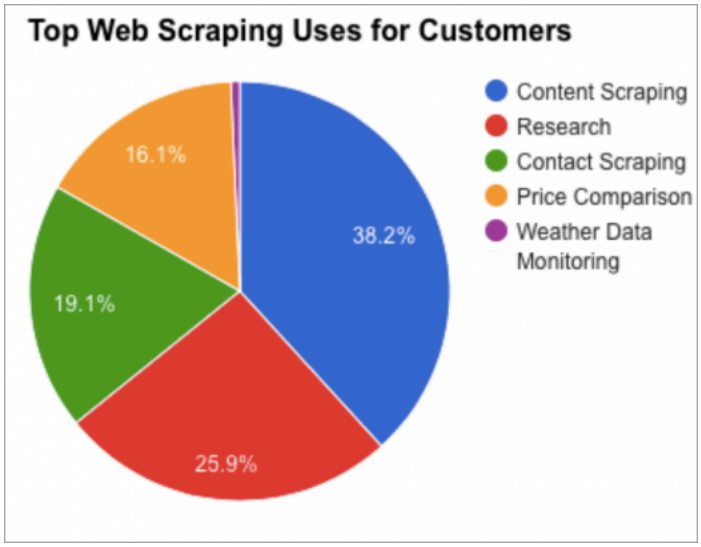
- **How is Web Scraping performed?**
- In order to scrape data from a website, software or a program is used. This program is called Scraper. This program sends a GET request to the website from which the data needs to be scrapped.

- As a result of this request, an HTML document which will be analyzed by this program is received. Then it makes a search for your required data and makes the conversion in the required format.

- There are two different methods for performing web scraping, one is accessing www via HTTP or a web browser and the second one is making use of bot or web crawler.

Uses of Web Scraping
Web Scraping is used for research work, sales, marketing, finance, e-commerce, etc. Many times, it is used to know more about your competitors.

The following image will show you the typical uses of web scraping and their percentage.



Comparison of Top Web Scraping Tools

Web scraping tools	Tagline	Output Formats	Users	Free Trial	Price
Smartproxy	Easy data gathering at scale with Web Scraping API	JSON, CSV, and XLSX	Individuals and Businesses	--	Pay As You Go: Starts at $12.50 Micro: $80.00 Starter: $225.00 Regular: $400.00
ScraperAPI	We handle 2 billion API requests per month for over 1,000 businesses and developers around the world	TXT, HTML CSV, or Excel formats	Small, medium, enterprise as well as individuals	Available	1000 free API calls Then starts with $29 per month only. (See Discount Below)
Web Scraper	Chrome extension: A free tool to scrape dynamic web pages.	CSV or through API, Webhooks, Dropbox.	--	Available	Free: Browser extension. Project: $50/month. Professional: $100/month. Business: $200/month. Scale: $300/month.
Grepsr	Web Scraping service platform that's effortless.	XML, XLS, CSV, & JSON	Every one.	You can sign up for free	Starter Plan: Starts at $129/site for 50K records. Monthly Plan: Starts at $99/site. Enterprise Plan: (Get a quote)
ParseHub	A web scraping tool that is easy to use.	JSON, Excel, and API.	Executives, Data Scientists, software developers, business analysts, pricing analysts, consultants, marketing professionals etc.	Free plan available.	Free plan for everyone. Standard: $149 per month, Professional: $499 per month, & Enterprise: Get a quote.

Benefits of Web Scraping

1. Web scrapers can automatically pull data from a few websites simultaneously, saving time and collecting more relevant information than one person can manually do.
2. Web scraping enables downloading and managing data on your local computer in spreadsheets or databases.
3. Web scrapers can be run on a schedule to get up-to-date data regularly and exported in the desired format.
4. Data accuracy is essential to any data-driven business. If you are looking for a data extraction tool that is human-free, hassle-free, and accurate, then web scraping is the answer.

The Disadvantages of Web Scraping

1. Because websites are constantly changing their HTML structure, sometimes scrapers can break. Whether you use web scraping software or write your own code, you should perform regular maintenance to keep your scrapers clean and working correctly.
2. The data collected needs to be properly read and understood to be processed, which can take a lot of time and effort.
3. Scraping large sites requires a vast number of requests. Some websites might block IP addresses from which many requests come in.
4. Many sites restrict access when requests come from certain countries, so that you will need proxy servers. And free or cheap proxies usually don't help in these cases because many people use them, and those IPs are already blocked.
5. Many modern websites render content when the page loads in the browser. When you try to view the page's source code or get it with a simple GET request, you will receive the message "You need to enable JavaScript to run this application". You will need headless browsers to scrape such sites with dynamic content. And with many pages, the rendering process requires more time and hardware resources.

What is API?

API stands for Application Programming Interface, which acts as an intermediary, allowing websites and software to communicate and exchange data and information.

To contact the API, you need to send it a request. The client must provide the URL and HTTP method to process the request correctly. You can add headers, body, and request parameters depending on the method.

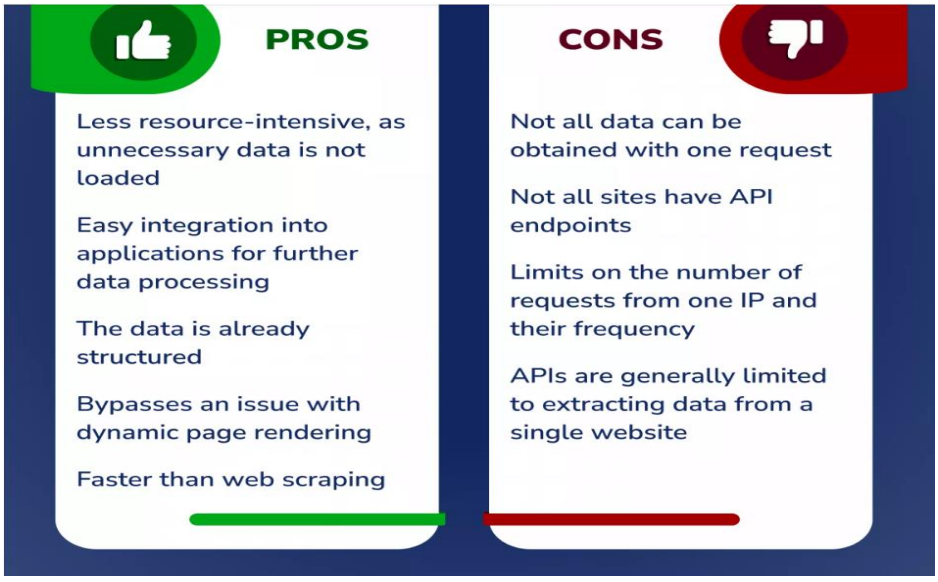
- Headers provide metadata about the request.
- The body contains data such as fields for a new row in a database.

The API will process the request and send the response received from the web server.

Endpoints work in conjunction with API methods. Endpoints are specific URLs that the application uses to communicate with third-party services and its users.

What is API Scraping?

API scraping is data collection by making requests to endpoints we found when analyzing the traffic between the web server and the site or application.



Benefits of API Scraping

1. You don't put an extra load on your hardware using this method to collect content.
2. API scraping can be implemented into a developer application with just a set of credentials.
3. The results are often provided in XML or JSON format, where the data is already structured and convenient for further processing.
4. API use helps to overcome problems like Javascript rendering and CAPTCHA avoidance.
5. If you need to collect hundreds, thousands, or millions of data, API solves this problem faster than web scraping.

The Disadvantages of API Scraping

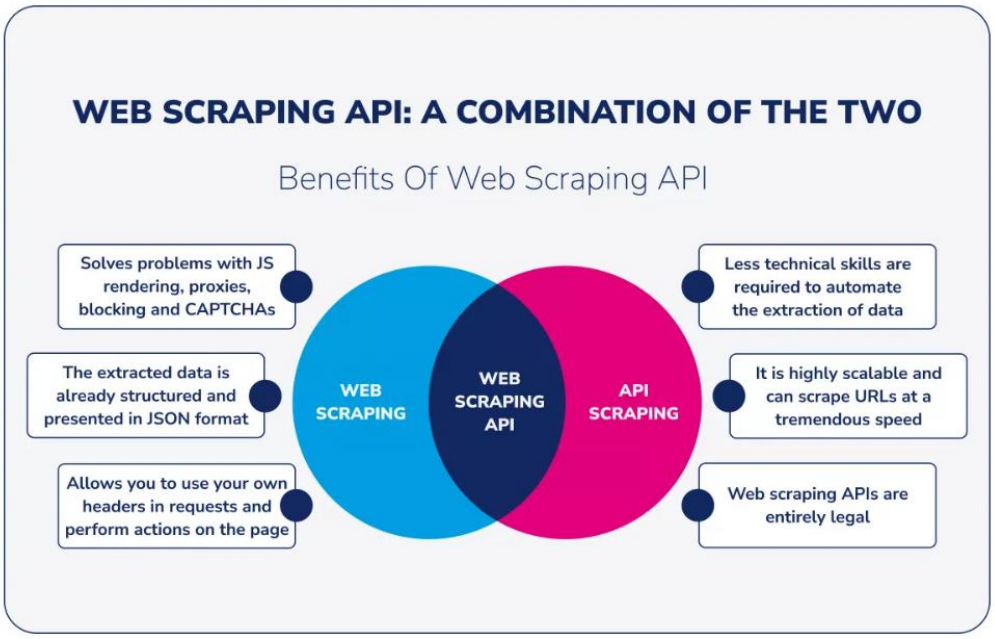
1. Not all data can be available by accessing a single endpoint due to the dataset available when accessing the endpoint predefined by the developer. So we may need to refer to several endpoints to collect the complete data set.
2. Not all sites have the ability to access the API because the server immediately returns only HTML with the page content.
3. The same API may limit the number of requests from one IP address and frequency.
4. APIs are generally limited to extracting data from a single website (unless they're aggregators), but with web scraping, you can get data from multiple websites. In addition, API lets you get only a specific set of data provided by the developers.

What is Web Scraping API?

Web scraping API is a powerful tool for data extraction from websites with the ability to rotate proxies, render JavaScript, bypass CAPTCHAs, and avoid blocking with a simple API call. You don't need to create a scraping application from scratch and take care of proxies, infrastructure maintenance, scaling, and so on. It is enough to make a request using the provided API and get the content of the needed web page. If necessary, you can optionally send

in a request the proxy country and type, custom headers, cookies, and waiting time, and even execute JavaScript in the request.

In other words, web scraping API connects the data extraction software built by the service provider with the websites you need to scrape.



Benefits of Web Scraping API

What is Web Scraping API Used For?

Web scraping API is used for various purposes like analytics, data mining, content aggregation, market research, and content marketing for better ranking in search engines. It can also extract specific data from any website or blog.

Companies use this tool when there is usually no time, specialists, or budget to develop their own solution, which needs to be supported and maintained.

Benefits of Web Scraping API

1. Solves problems with JavaScript rendering, proxies, blocking and CAPTCHAs.
2. The extracted data is already structured and presented as a rule in JSON format.

3. Web scraping API allows you to use your own custom headers (user agents, cookies, etc.) when making requests to a website in a simple way.
4. It can be used by anyone who wants to autonomously automate the tasks associated with scraping content from the web.
5. Most web scraping API services are built with scalability, meaning they can scrape URLs at tremendous speed, often scanning thousands of pages per second and retrieving data daily.
6. Web scraping APIs are entirely legal. However, it is better to respect the site owners and not scrape sites very quickly, as sites may not be designed for a large number of requests.

Data Extraction Process with Web Scraping API

1. To gather data, simply use the base API endpoint and add the URL you want to scrape as the body parameter and your API key as the header.

   There are also some optional parameters that you can choose. These include custom titles, the usage of rotating proxies, their type and country, blocking images and CSS, timeouts, browser window sizes, and JS scenarios, such as filling out a form or clicking a button.

2. Send the extracted data to your own tools for further HTML processing, for example, for parsing using regular expressions and obtaining specific data in a structured form.

   Our service allows you to use extraction rules to get only the data you need in JSON format without the need to save the raw data.

3. Stream data to your database. You can use your own software tools or integration platforms such as Zapier or Make. In the article about Google Maps' no-code scraping, we wrote about this in more detail.

Feature Selection & Feature Extraction

In this notebook, we're going to learn about 2 SKLearn modules. They're

1. Feature Selection
2. Feature Extraction

Feature Selection:- This module is used for feature selection/dimensionality reduction on given datasets. This is done either to improve estimators' accuracy scores or to boost their performance on very high-dimensional datasets.

Feature Extraction:- This module is used to extract features in a format supported by machine learning algorithms from the given datasets consisting of formats such as text and image.

The main difference:- Feature Extraction transforms an arbitrary data, such as text or images, into numerical features that is understood by machine learning algorithms. Feature Selection on the other hand is a machine learning technique applied on these (numerical) features.

Let's try to understand the different classes under these 2 modules with the help of the Abalone dataset.

As always, I'll keep this notebook well commented & organized for easy understanding. Please do consider it to UPVOTE if its helpful.

Feature Selection

Feature selection is one of the important concepts of machine learning, which highly impacts the performance of the model. As machine learning works on the concept of "Garbage In Garbage Out", so we always need to input the most appropriate and relevant dataset to the model in order to get a better result.

What is Feature Selection?

A feature is an attribute that has an impact on a problem or is useful for the problem, and choosing the important features for the model is known as feature selection. Each machine learning process depends on feature engineering, which mainly contains two processes; which are Feature Selection and Feature Extraction. Although feature selection and extraction processes may have the same objective, both are completely different from each other. The main difference between them is that feature selection is about selecting the subset of the original feature set, whereas feature extraction creates new features. Feature selection is a way of reducing the input variable for the model by using only relevant data in order to reduce overfitting in the model.

So, we can define feature Selection as, "***It is a process of automatically or manually selecting the subset of most appropriate and relevant features to be used in model building***." Feature selection is performed by either including the important features or excluding the irrelevant features in the dataset without changing them.

Need for Feature Selection

Before implementing any technique, it is really important to understand, need for the technique and so for the Feature Selection. As we know, in machine learning, it is necessary to provide a pre-processed and good input dataset in order to get better outcomes. We collect a huge amount of data to train our model and help it to learn better. Generally, the dataset consists of noisy data, irrelevant data, and some part of useful data. Moreover, the huge amount of data also slows down the training process of the model, and with noise and irrelevant data, the model may not predict and perform well. So, it is very necessary to remove such noises and less-important data from the dataset and to do this, and Feature selection techniques are used.

Selecting the best features helps the model to perform well. For example, Suppose we want to create a model that automatically decides which car should be crushed for a spare part, and to do this, we have a dataset. This dataset contains a Model of the car, Year, Owner's name, Miles. So, in this dataset, the name of the owner does not contribute to the model performance as it does not decide if the car should be crushed or not, so we can remove this column and select the rest of the features(column) for the model building.

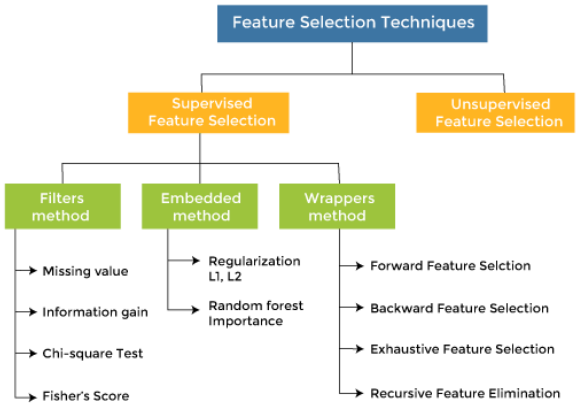
Below are some benefits of using feature selection in machine learning:

- **It helps in avoiding the curse of dimensionality.**
- **It helps in the simplification of the model so that it can be easily interpreted by the researchers.**
- **It reduces the training time.**
- **It reduces overfitting hence enhance the generalization.**

Feature Selection Techniques

There are mainly two types of Feature Selection techniques, which are:

- **Supervised Feature Selection technique**
  Supervised Feature selection techniques consider the target variable and can be used for the labelled dataset.
- **Unsupervised Feature Selection technique**
  Unsupervised Feature selection techniques ignore the target variable and can be used for the unlabelled dataset.



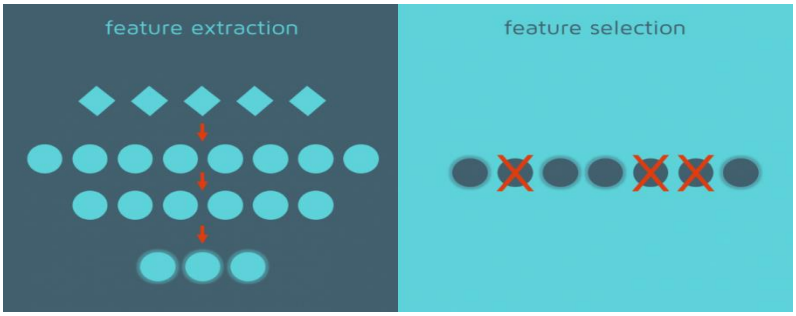
Feature Generation

Before we get into the details let's review what a feature is. A feature (or column) represents a measurable piece of data like name, age or gender.It is the basic building block of a dataset. The quality of a feature can vary significantly and has an immense effect on model performance. We can improve the quality of a dataset's features in the pre-processing stage using processes like Feature Generation and Feature Selection.

Feature Generation (also known as feature construction, feature extraction or feature engineering) is the process of transforming features into new features that better relate to the target. This can involve mapping a feature into a new feature using a function like log, or creating a new feature from one or multiple features using multiplication or addition.

Feature Generation can improve model performance when there is a feature interaction. Two or more features interact if the combined effect is (greater or less) than the sum of their individual effects. It is possible to make interactions with three or more features, but this tends to result in diminishing returns.

Feature Generation is often overlooked as it is assumed that the model will learn any relevant relationships between features to predict the target variable. However, the generation of new flexible features is important as it allows us to use less complex models that are faster to run and easier to understand and maintain.



Examples of Feature Generation techniques

A transformation is a mapping that is used to transform a feature into a new feature. The right transformation depends on the type and structure of the data, data size and the goal. This can involve transforming single feature into a new feature using standard operators like log, square, power, exponential, reciprocal, addition, division, multiplication etc.

Often the relationship between dependent and independent variables are assumed linear, but this is not always the case. There are feature combinations that cannot be represented by a linear system. A new feature can be created based on a polynomial combination of numeric features in a dataset. Moreover, new features can be created using trigonometric combinations.

Manual vs Automated feature generation

Feature Generation was an ad-hoc manual process that depended on domain knowledge, intuition, data exploration and creativity. However, this process is dataset-dependent, time-consuming, tedious, subjective, and it is not a scalable solution. Automated Feature Generation automatically generates features using a framework; these features can be filtered using Feature Selection to avoid feature explosion. Below you can find some popular open source libraries for automated feature engineering:

Pycaret – PyCaret

Featuretools for advanced usage Home What is Featuretools? — Featuretools 1.1.0 documentation

Optuna –A hyperparameter optimization framework

Feature-engine A Python library for Feature Engineering for Machine Learning — 1.1.2

ExploreKit GitHub – giladkatz/ExploreKit

Conclusion

Feature Generation involves creating new features which can improve model accuracy. In addition to manual processes, there are several frameworks that can be used to automatically generate new features that expose the underlying problem space. These features can subsequently be filtered using Feature Selection, to ensure that only a subset of the most important features is used. This process effectively reduces model complexity and improves model accuracy as well as interpretability.

FILTERING THEORY AND ALGORITHM

The Kalman filtering algorithm is well known as the procedure to estimate the optimal state vector in dynamic system with stochastic properties. On the other hand, an algorithm based on the projection filter for a stochastic dynamic system which corresponds to the Kalman filtering algorithm based on the Wiener filter was presented as the inverse analysis method. In this paper, the parametric projection filtering algorithm is employed in place of above both filters as the third filter.

The dynamic system considered in the inverse problem can be given by the following discrete time model:

State equation:

$$z_{t+1} = \Phi_t z_t + \Gamma_t \zeta_t \tag{1}$$

Observation equation:

$$y_t = M_t z_t + v_t \tag{2}$$

where $\mathbf{z}_t$, $\mathbf{y}_t$, ζ_t and v_t are the state vector, the observation vector, the system noise vector and the observation noise vector, respectively, and Φ_t, Γ_t, and $\mathbf{M}_t$ are the state transition matrix, the driving matrix and the observation matrix, respectively. The noise vectors are assumed to possess the following stochastic properties.

$$\begin{aligned} &E\zeta_t = 0, E\zeta_t\zeta_e^T = S_t\delta_{te}, Ez_t\zeta_t^T = 0 \\ &Ev_t = 0, Ev_t v_e^T = Q_t\delta_{te}, Ez_t v_e^T = 0 \end{aligned} \tag{3}$$

where E denotes expectation and δ is Kronecher's delta.

The parametric projection filtering algorithm can be constructed with the recursive procedure as follows,

Filter equation:

$$\hat{z}_{t/t} = \hat{z}_{t/t-1} + B_t y_t - M_t \hat{z}_{t+1/t} \tag{4}$$

Filter gain for parametric projection filter:

$$B_t = M_t^T M_t M_t^T + \gamma Q_t^{-1} \quad (5)$$

Initial condition:

$$\hat{z}_{0/-1} = \bar{z}_0 z_0 = E z_0 \quad (6)$$

As seen above equation (5), the parametric projection filter does not depend on the estimation error covariance matrix in contrast with the Kalman filter. It is well known that the inverse analysis is attended with unstable calculation generally. It is considered that the parametric projection filter is including the parameter γ to regulate the stability of iterative calculations.

1. The most common **filter** is a software filter that reads data and manipulates the data to fit another output pattern or removes data that may not be needed. For example, spam filters help filter unwanted e-mails from reaching your Inbox.

2. Hardware devices can also be **filters**. For example, a firewall can filter network traffic to help protect a network.

3. The term **filter** also refers to a device inside a power supply that smooths out pulsing DC (direct current).

4. With Adobe Photoshop, a **filter** is a digital effect used to modify images. See our Photoshop filters page for a complete listing of Photoshop filters.

5. Alternatively called **censorware**, a **swear filter**, or a **content filter**, a **filter** can describe software or hardware which selectively blocks specific data. For example, a parental control Internet filter can block obscene websites from children on the Internet.

Wrapper methods

In *wrapper methods,* the *feature selection* process is based on a specific machine learning algorithm that we are trying to fit on a given dataset.

It follows a *greedy search approach* by evaluating all the possible combinations of features against the *evaluation criterion*. The *evaluation criterion* is simply the performance measure which depends on the type of problem, for e.g. For *regression* evaluation criterion can be p-values, R-squared, Adjusted R-squared, similarly for *classification* the evaluation criterion can be accuracy, precision, recall, f1-score, etc. Finally, it selects the combination of features that gives the optimal results for the specified machine learning algorithm.

Filter methods	Wrapper methods	Embedded methods
Generic set of methods which do not incorporate a **specific machine learning algorithm.**	Evaluates on a **specific machine learning algorithm** to find optimal features.	Embeds (fix) features during **model building process.** Feature selection is done by observing each iteration of model training phase.
Much **faster** compared to Wrapper methods in terms of time complexity	**High computation time** for a dataset with many features	Sits **between Filter methods and Wrapper methods** in terms of time complexity
Less prone to **over-fitting**	High chances of **over-fitting** because it involves training of machine learning models with different combination of features	Generally used to reduce **over-fitting** by **penalizing** the coefficients of a model being too large.
Examples – **Correlation, Chi-Square test, ANOVA, Information gain** etc.	Examples - **Forward Selection, Backward elimination, Stepwise selection** etc.	Examples - **LASSO, Elastic Net, Ridge Regression** etc.

What Is a Decision Tree?

A decision tree is a type of supervised machine learning used to categorize or make predictions based on how a previous set of questions were answered. The model is a form of supervised learning, meaning that the model is trained and tested on a set of data that contains the desired categorization.

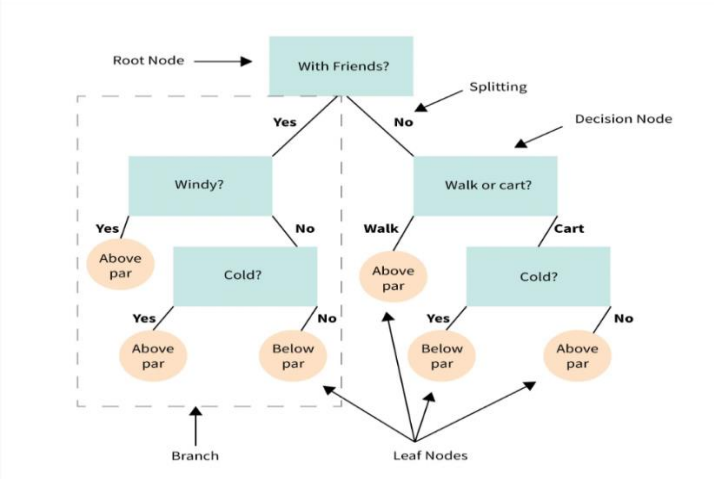
The decision tree may not always provide a clear-cut answer or decision. Instead, it may present options so the data scientist can make an informed decision on their own. Decision trees imitate human thinking, so it's generally easy for data scientists to understand and interpret the results.

How Does the Decision Tree Work?

Before we dive into how a decision tree worksExternal link:open_in_new, let's define some key terms of a decision tree.

- **Root node:** The base of the decision tree.
- **Splitting:** The process of dividing a node into multiple sub-nodes.
- **Decision node:** When a sub-node is further split into additional sub-nodes.
- **Leaf node:** When a sub-node does not further split into additional sub-nodes; represents possible outcomes.
- **Pruning:** The process of removing sub-nodes of a decision tree.
- **Branch:** A subsection of the decision tree consisting of multiple nodes.

A decision tree resembles, well, a tree. The base of the tree is the root node. From the root node flows a series of decision nodes that depict decisions to be made. From the decision nodes are leaf nodes that represent the consequences of those decisions. Each decision node represents a question or split point, and the leaf nodes that stem from a decision node represent the possible answers. Leaf nodes sprout from decision nodes similar to how a leaf sprouts on a tree branch. This is why we call each subsection of a decision tree a "branch." Let's take a look at an example for this. You're a golfer, and a consistent one at that. On any given day you want to predict where your score will be in two buckets: below par or over par.



Decision Tree Variables and Design

In the golf example, each outcome is independent in that it does not depend on what happened in the previous coin toss. Dependent variables, on the other hand, are those that are influenced by events before them.

Building a decision tree involves construction, in which you select the attributes and conditions that will produce the tree. Then, the tree is pruned to remove irrelevant branches that could inhibit accuracy. Pruning involves spotting outliers, data points far outside the norm, that could throw off the calculations by giving too much weight to rare occurrences in the data.

Maybe temperature is not important when it comes to your golf score or there was a day when you scored really poorly that's throwing off your decision tree. As you're exploring the data for your decision tree, you can prune specific outliers like your one bad day on the course. You can also prune entire decision nodes, like temperature, that may be irrelevant to classifying your data.

Well-designed decision trees present data with few nodes and branches. You can draw a simple decision tree by hand on a piece of paper or a whiteboard. More complex problems, however, require the use of decision tree software.

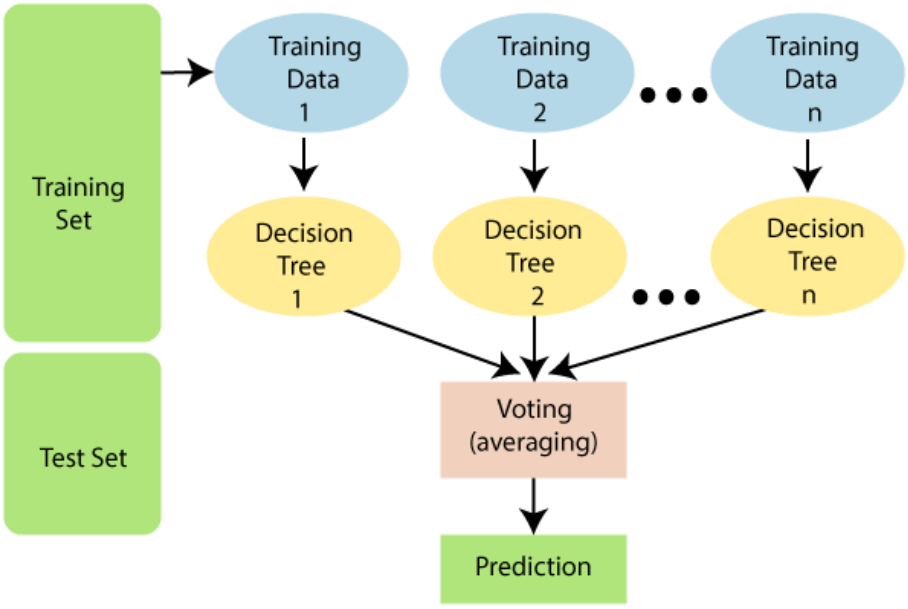
Random Forest Algorithm

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

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

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

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



Assumptions for Random Forest

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

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

Why use Random Forest?

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

<=”” li=””>

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

How does Random Forest algorithm work?

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

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

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

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

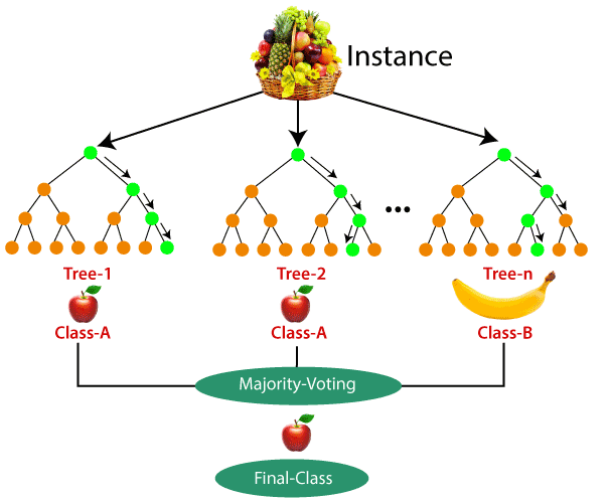
Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

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

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

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



Applications of Random Forest

There are mainly four sectors where Random forest mostly used:

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

Advantages of Random Forest

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

Disadvantages of Random Forest

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