Machine Learning

Author[1]

January 6, 2025

[1]www.example.com

Contents

List of Figures

List of Tables

1

Introduction to Machine Learning

Artificial Intelligence (AI) in nutshell is, *enable computers to think*. AI can simulate human thinking capability and behavior. **Machine learning** (ML)is a subfiled of AI, that allows machines to learn from data without being programmed explicitly.

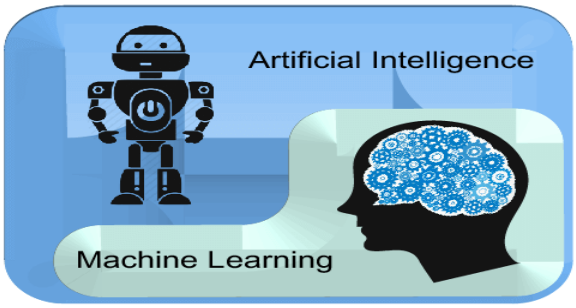


Figure 1.1: AI vs ML

ML can automatically detect patterns in data that human brain or traditional methods finds very difficult. Today we enjoy reliable web search engines results, robust email spam filters, convenient text and voice recognition software, challenging chess-playing programs etc. All runs machine learning algorithms.

Data Science (DS) is all about extracting useful insights from the data by processing that data using different steps by using tools, statistical models & Machine learning algorithms. The isights pulled from data is later used to drive business outcomes.

Deeplearning (DL) is a subfield of machine learning. DL is based on deep nearual networks inspired from how human brain functions. DL is a fast evolving area.

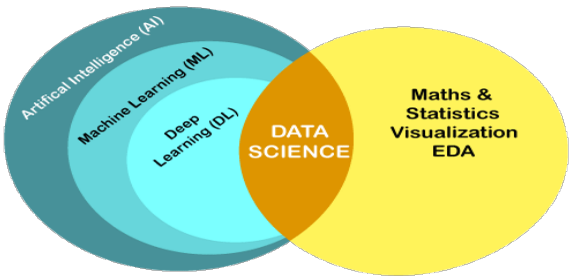


Figure 1.2: Artificial Intelligence Subfields

Table 1.1: AI vs ML

AI	ML
Goal is to make a smart computer system like humans to solve complex problems wide range of scope	Goal is to allow machines to learn from data so that they can give accurate output limited scope
The main applications are Siri, customer support using catboats, Expert System, Online game playing, intelligent humanoid robot, etc.	The main applications are Online recommender system, Google search algorithms, Facebook auto friend tagging suggestions, etc.
It includes learning, reasoning, and self-correction	It includes learning and self-correction when introduced with new data
Deals with Structured, semi-structured, and unstructured data	Deals with Structured and semi-structured data

1.1 Dataset

In machine learning, an unknown **universal dataset** is assumed to exist that contains all the possible data pairs as well as their probability distribution. While in real applications, what we observe is only a subset due to the lack of memory or some unavoidable reasons. This acquired dataset is called the **training data** and is used to learn the properties and nature of the universal dataset.

There are two types of datasets used in machine learning, **labeled dataset** and **unlabeled dataset**. Each dimension of a vector in a dataset is called an **attribute**, **feature**, **variable** or **element**.

- **Labelled Dataset D :** $X = \{x^{(n)} \in \mathbb{R}^d, Y^{(n)} \in \mathbb{R}\}_{n=1}^{N}$
- **Unlabelled Dataset D :** $X = \{x^{(n)} \in \mathbb{R}^d\}_{n=1}^{N}$

Where X denotes the feature set & Y stands for the label set.

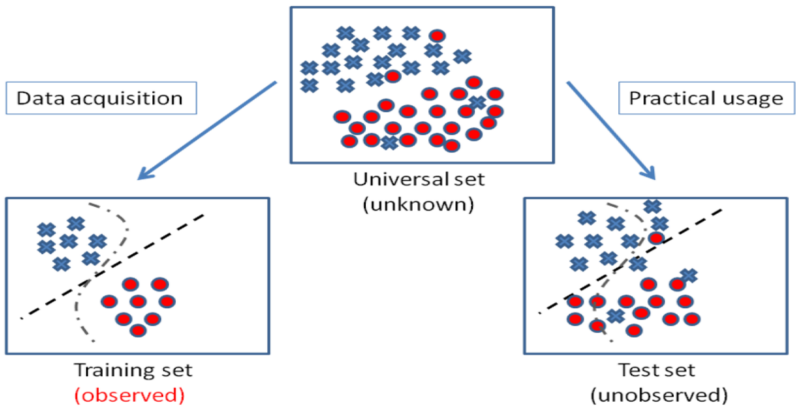


Figure 1.3: Universal Dataset

- The first property a machine can learn in a labeled data set is a **separating boundary** or **decision boundary**.

1.2 No Free Lunch Rule

For machine learning algorithms to work on test set, there are some rules to get satisfied. These rules are called the "**no free lunch rules**" and are defined on both train, test datasets and the properties to learn.

- On dataset, training dataset and the test dataset needs to be from the same distribution (same universal set).
- On the properties, the no free lunch rules ask the data scinetist to make assumptions on what property to learn and how to model the property.
  - If the separating boundary in a labeled dataset is desired, we also need to define the type of the boundary (ex. a straight line or a curve).
  - If we want to estimate the probability distribution of an unlabeled dataset, the distribution type should also be defined (ex. Gaussian distribution, Poison distribution etc.).

1.3 Ideals of Machine Learning

For a machine to make choices, the intuitive way is to model the problem into a mathematical expression.

- There are some cases where the mathematical expression could directly be designed from the problem background.

  - The vending machine could use the standards and security decorations of currency to detect false money. The set of hard coded rules can be used, so the solution is **designed** to detect false money.

- While in some other problems that we can only acquire several measurements and the corresponding labels, but do not know the specific relationship among them

  - **Learning** will be a better way to find the underlying connection. eg: Detect an email is spam or not. Different persons will write emails in different way, so the hardcoded rules won't work.

ML involves the techniques and basis from both statistics and computer science.

- **Statistics**: Learning and inference the statistical properties from given data

- **Computer science**: Efficient algorithms for model representation, objective function optimization, and performance evaluation. eg: Tree data structure for decision trees, GPU based architectures for deep learing algorithms, dynamic programming idea to solve backpropagation algorithms efficiently.

Two key elements of Machine learning are **Modeling** & **Optimization**.

- We model the separating boundary or probability distribution of the given training set

- Optimization techniques are used to seek the best parameters of the chosen model

The knowledge acquired from human understandings is called the **domain knowledge** and the knowledge learned from a given training set is called **data driven knowledge**.

1.4 Types of Machine Learning

Three types of machine learning tecniques are shown in Fig. 1.4

Fig. 1.5 further shows the idea of supervised learning.

- An unknown target function f (or target distribution) that maps each feature vector in the universal dataset to its corresponding label is assumed to exist

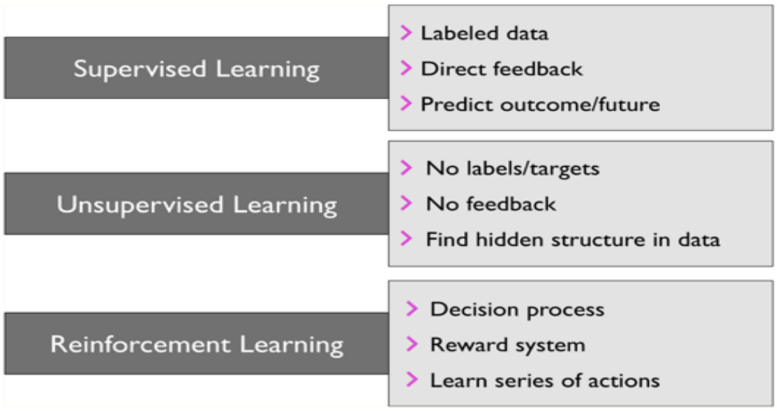


Figure 1.4: Types of Learning

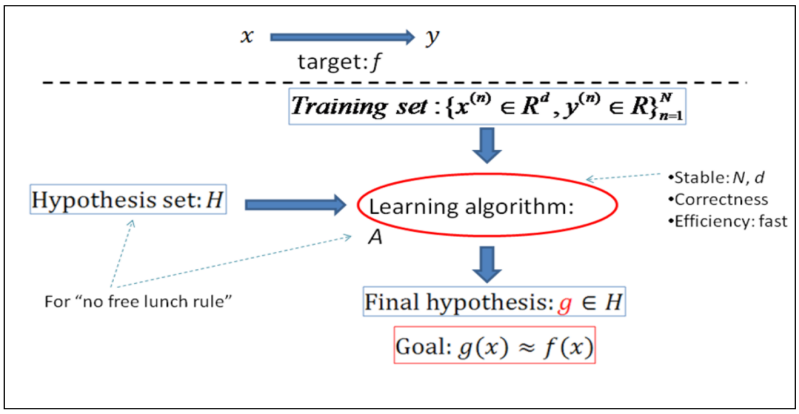


Figure 1.5: Supervised Learning Structure

- Because we do not have any idea about f (that looks like a linear boundary or a circular boundary?), a **hypothesis** set H is necessary to be defined that contains several hypotheses $'h'$s.
- The goal of supervised learning is to find the best h, called the **final hypothesis** g that best approximate the target function f
- To find the final hypothesis g, we need a **learning algorithm** A, that includes the **objective function** which needs to be optimized for searching g
- There are different **optimization methods** for different algorithms like Graidient descent, Maxium Margine Hyperplane etc.
- The hypothesis set H and the objective function jointly model the property to learn for the no free lunch rules

In supervised learning the desired goal is to find g that has the lowest error rate for classifying data generated from f. When the error rate is defined on the training set, it is named the **in-sample error** $E_{in}(h)$, while the error rate on the universal set or more practically the (unknown set or reserved set) is **out-of-sample error** $E_{out}(h)$.

Based on these definitions, the desired final hypothesis g is the one that achieves the lowest $E_{out}(h)$ over the whole hypothesis set

$$g = \arg\min_{h} E_{out}(h) \tag{1.1}$$

A supervised learning strategy called **empirical risk minimization** (ERM) is proposed to achieve low $E_{out}(g)$ by minimizing $E_{in}(g)$ during the training step.

$$g = \arg\min_{h} E_{in}(h) \tag{1.2}$$

An objective function like $E_{in}(h)$ can be separated into a **loss function** and a **penalty function**.

- The loss function measures the classification error defined on the training set
- The penalty function gives each hypothesis a priority

In **unsupervised learning**, goal is to discover *interesting structure* in the data; this is sometimes called **knowledge discovery**. Unlike supervised learning, we are not told what the desired output is for each input.

2

Linear Algebra & Calculus

Linear algebra provides us mathematical concepts which are the foundations of many machine learning models. The basic elements in linear algrebra are:

- Scalar : A number eg: 8
- Vector, Row vector, Column vector, point , Array of numbers eg: [1,4,7] is a 3D vector. By default vector is a column vector.
- Matrix : An array of vectors
- Tensors : Higher order matrices are called tensors. A tensor of order 3 will have 3 matrices stacked together. eg : RGB images, each R,G,B channel is a matrix.
  - Scalar : Tensor of order 0
  - Vector : Tensor of order 1
  - Matrix : Tensor of order 2
  - RGB image : Tensor of order 3
  - A batch of RGB imgages : Tensor of order 4

2.1 Vector Opearations

Let $x = [x_1 + x_2 + \cdots + x_n]$, $y = [y_1 + y_2 + \cdots + y_n]$ be two vectors. Distance between a vector and origin or with another vectors can be computed as follows

$$d(x, 0) = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \tag{2.1}$$

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} \tag{2.2}$$

There are two types of vector multiplications

- Dot product $(a.b)$ or Scalar product : This results in scalar quantity.
- Cross product $(a \times b)$: This results in result in vector quantity.

$$a.b = a_1b_1 + a_2b_2 + a_3b_3 + \cdots + a_nb_n \tag{2.3}$$
$$= a^T b \tag{2.4}$$
$$= \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta \tag{2.5}$$

The angle between two vectors can be computed uinsg Eq. 2.6

$$\theta = \arccos\left(\frac{a.b}{\|\mathbf{a}\| \, \|\mathbf{b}\|}\right) \tag{2.6}$$

The **projection of a vector** a on another vector is given in Eq. 2.7 and also shown in Fig. 2.1

$$\frac{a.b}{\|\mathbf{b}\|} = \|\mathbf{a}\| \, . \cos\theta \tag{2.7}$$

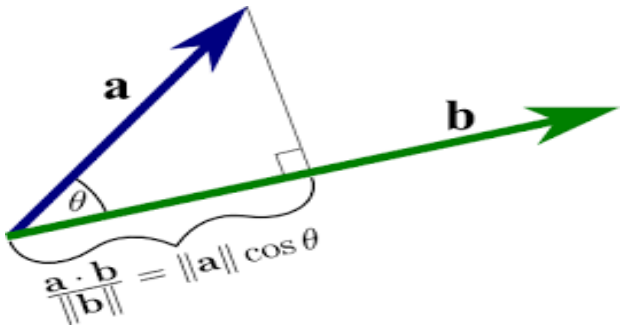


Figure 2.1: Projection of a vector

A **unit vecor** of of vector a is a vector which has the following properties:

1. Same direction as that of vector a
2. $\|\mathbf{a}\| = 1$

The equation of a line in 2D can be written as Eq. 2.8 and is called general equation of a line.

$$a_1x_1 + a_2x_2 + d = 0 \tag{2.8}$$

The **slope intecept form** of a line can be written as Eq. 2.9, whre m is the **slope** and c is the **intercept**. This formula, we can get by rewriting the general equation of line.

$$x_2 = mx_1 + c \tag{2.9}$$

2.2 Generalization

In linear algebra many equations can be easily generalized to higher dimensions as given in Table 2.1.

Table 2.1: Generalization

2D	3D	nD
line $a_1x_1 + a_2x_2 + d = 0$	plane $a_1x_1 + a_2x_2 + a_3x_3 + d = 0$	Hyperplane : $a_1x_1 + a_2x_2 + a_3x_3 + \cdots +$
Circle: $x_1^2 + x_2^2 + d = 0$	Sphere: $x_1^2 + x_2^2 + x_3^2 + d = 0$	Hypersphere: $x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2$
Ellipse: $\frac{x_1^2}{a_1^2} + \frac{x_2^2}{a_2^2} - 1 = 0$	Ellipsoid: $\frac{x_1^2}{a_1^2} + \frac{x_2^2}{a_2^2} + \frac{x_3^2}{a_3^2} - 1 = 0$	Hyper ellipsoid: $\frac{x_1^2}{a_1^2} + \frac{x_2^2}{a_2^2} + \cdots + \frac{x_n^2}{a_n^2} -$
Rectangle	Cuboid	Hyper Cuboid
Square	Cube	Hypercube

2.3 Half spaces and Decision Boundary

The equation of a hyperplane can be considered as a decision bounrary and can be written as Eq. 2.10. This can be written as vector dot product as Eq. 2.11, where w_0 is the intercept. When the line/ hyper plane pasess throguh the origin $w_0 = 0$.

$$w_0 + \sum_{i=1}^{N} w_i x_i = 0 \tag{2.10}$$

$$w_0 + W^T X = 0 \tag{2.11}$$

- Planes of any dimensions are typically denoted as π.
- A line or hyperplane divides the space into two **half spaces**. The first half lies above and the other half lies below.

Thus equation of a line passing through origin can be written as Eq. 2.12.

- $W^T X = 0$, implies $\cos\theta = 0$ and hence $\theta = 90$, i.e W is perpendicular to the plane.
- If we take any point x in the plane, we get the dot product value with W as 0, hence the equation.

$$W^T X = 0 \tag{2.12}$$

$$\|\mathbf{W}\| \, \|\mathbf{X}\| \cos\theta = 0 \tag{2.13}$$

The distance of a point to the plane can be computed by taking the projection of point on the normal to the plane. This is shown in Fig. 2.2.

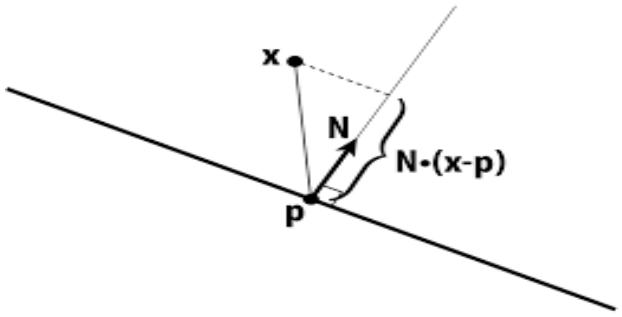


Figure 2.2: Distance of a point from plane

- If we replace W by a unit vector $\hat{W}$, we still get the dot product 0.
- The plane can also be represnted by $\hat{W}$ passing through origin.
- For any point in the space, $W^T X$ can take +ve, -ve or 0 value depending on the value of θ. Sign gives information about the half space in which the point belongs.

$$\theta = \begin{cases} 0 & \text{point lies in the plane} \\ < 90 & \text{dot product wiil be +ve and the point lies in the same side of the normal} \\ > 90 & \text{dot product will be -ve and the point lies in the opposite side of the normal} \end{cases}$$

2.4 Vector Calculus : Gradient

Many optimization problems in machine learning like- finding the best hyperplane that separates two clasess, mimimization of cost functions are solved using the idea of differentiations. We use scalar and vector differentiations extensively to solve the optimization problems.

- The differentiation has two meanings:
  - Geometrical meaning - **Slope of the tangent** $\tan\theta$. When a line touches exactly one point in the curve we call it as **tangent** to the curve.
  - Physical meaning - **Rate of change** - How much does y changes as x changes.
- **Derivatives** : differentiation done on scalar

- **Gradient** : differentiation done on vector. We use gradient methods for finding optimal weights for ML algorithms like neural network and for optimization of various cost functions.

2.4.1 Gradient Descent

Solving a maximization or minimization probelm on a scaler variable or variable with just one dimension is done by following the below steps. eg: find the value of x for which the function $y = x^2$ has minimum value?

- Find the derivative of the function $y = x^2$ with respect to x
- Equate the derivative to 0 to find critical points. $2x = 0 => x = 0$. so $x = 0$ may be a point of minimum or maximum.
- To decide on point of minimum or maximum, take the second derivative of $y = x^2$, we get 2 which is +ve , so its a point of minimum.
- if second defivative is
  - +ve : critical point is point of minimum
  - -ve : critical point is point of maximum
  - 0 : critical point is point of inflation -further tests are rquired to conclude on point of minimum or maximum.

Solving maxima, minima is not easy with gradients i.e for vectors where we have more than one dimension to optimize. To solve the optimization on vectors, we can use gradient descend / ascend method. We start with a random value and using gradient descend / ascend method we moves towards minima / maxima in an iterative manner.

- Maximization method can be easily switched to minimization and vice versa. Maximization function $f(x)$ is equivallent to minimization of $-f(x)$ and vice versa.
- At minima slope changes its sign from negative to positve.
- At maxima slope changes its sign from positve to negative.

In gradient descent method for optimizing the weight w, the value of w at current iteration t is computed using Eq. 2.14, where $r > 0$ is called the **learning rate**.

$$w_t = w_{t-1} - r * \frac{\mathrm{d}f(w,x)}{\mathrm{d}w}_{w_{t-1}} \tag{2.14}$$

- As we move towards optimum value, gradient descent / ascent makes smaller changes as the slope approches 0.

- There is a chance of oscillation around optimum point with fixed value of r. eg: for inimizing parabola curve $y = x^2$ with $r = 1$, the function oscillates between $x = -0.5$ & $x = 0.5$.
- The solution to avoid such possible oscillation is to set r as a variable which varies based on the iteration number t. $r = h(t)$ such that as $t \uparrow r \downarrow$.

Let us solve the optimization probelm for linear regression using gradient descend. Recall that the objective function of linear regression is $L(w) = \arg\min_{w,w_0} \sum_{i=1}^{N}(y_i - w^T x_i + w_0)^2$. Let us assume w_0 is 0 for simplicity and we get our objective function as Eq. 2.15

$$L(w) = \arg\min_{w} \frac{1}{N} \sum_{i=1}^{N}(y_i - w^T x_i)^2 \tag{2.15}$$

The gradient of this function with respect to weight w becomes

$$\nabla_w(L) = \frac{1}{N} \sum_{i=1}^{N}(-2x_i)(y_i - w^T x_i)$$

So the weight updating formula becomes

$$w_t = w_{t-1} - r * \frac{1}{N} \sum_{i=1}^{N}(-2x_i)(y_i - w_{t-1}^T x_i) \tag{2.16}$$

The dataset we use to train the model can have millions of records. Computing the summation part of the above equation is time consuming and requires us to load all the training data into memory. The solution to this problem is **mini-batch stochastic gradient descent (mini-batch SGD)**.

- Instead of using all data points in the training dataset, mini-batch SGD picks a *batch of size* $k << N$ in each iteration randomly (stocahstic) and update the weights.
- It can be proved that $\hat{w}_{batch-SGD} \approx \hat{w}_{GD}$. So the solution is as good as gradient descend. To summarize the terminologies,
  - $k = 1$: stocahstic gradient descend
  - $1 < k < N$: mini-batch SGD
  - $k = N$: gradient descend

2.5 L1 regularization and sparsity

Regularization is one of the technique used to find sweet spot in the bias variance curve. There are two regularization methods used namely L_1 regularization & L_2 regularization.

- L_1 regularization creates sparsity in feature vector (unimportant features get zeroed-out). This would be helpful in eliminating some of the features that are less important.

The reason why L_1 regularization creates sparsity is explained below. The L_1, L_2 regularization based optimization functions can be written as follows:

$$L_1 formula = loss + \lambda ||w||_1 \tag{2.17}$$

$$L_2 formula = loss + \lambda ||w||_2^2 \tag{2.18}$$

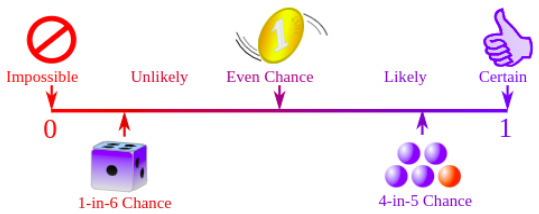
In the above equations the $loss$ and λ values are same for both optimization equations and λ is a constant. Let $w =< w_1, w_2, \ldots, w_d >$ be the weight vector of d dimesnions. The intutive idea of why L_1 regularizer creates sparsity can be understood as follows:

- The individual components of a weight vectors are not multiplied among themself. Therefore we can take the derivative independently using additive formula.
- When we take the derivative of L with respect to a particular dimension w_k:
  - $\frac{\partial L_1}{\partial w_k} = \pm 1$
  - $\frac{\partial L_2}{\partial w_k} = 2w_k$
- Derivative of L_2 is linear function but derivative of L_1 is a constant.
- As we approach the minima point, L_2 derivative value also decreases and almost approaches 0, but L_2 derivative $= \pm 1$, so L_1 succeeds in zeroing out the feature weights.

3

Probability and Statistics

What is the chance that my football team will win the premiership this year?. The **probability** denote the chance of occurence of an event. Probability value lies in interval $[0, 1]$ or $[0\%-100\%]$. A variable which takes the outcome of a random experiment is known as the **Random variable**. The columns of structured data in machine learning represents a set of random variables. There are two types of random variable, **Continuous random variable** (eg: income) & **Discrete random variable** (eg: Gender). The universal set of values for a random variable is called **population** which is practically not available to us in many cases, what we observe in real life is the subset of population called **sample**. An ideal sample maintains the proportions in the population.



Probability is always between 0 and 1

Random variable is represented by capital letters like X, Y. $P(X = x_1)$ or simply $P(x_1)$ represents the probability that the random variable X takes the value x_1 eg: P(Gender='M'). Mean of population is denoted as μ and mean of sample is denoted as $\bar{h}$. As we increase the size of sample size, the $\bar{h}$ approaches μ.

When we take the salary example, there are various designations in an organization. If we want to check all the salary brackets available in an organization then we are looking for all possible outcomes of random variable salary, this is called a **sample space** of salary. **Event** is a subset of sample space. eg: Salary of senior engineers.

Suppose in a retail store a customer has purchased items from counter desk 1 and desk 2. What is the probability that he has purchased 2 items from desk 1 where we already know that he has purchased less than 6 items in total? Here the event of total purchased items < 6 has already happened and we want to know what is the chance that a customer purchased 2 items from desk 1?.

+		D2					
		1	2	3	4	5	6
D1	1	2	3	4	5	6	7
	2	3	4	5	6	7	8
	3	4	5	6	7	8	9
	4	5	6	7	8	9	10
	5	6	7	8	9	10	11
	6	7	8	9	10	11	12

Figure 3.1: Conditional Probability

This chance for the above event can be computed using **Conditional probability** which is denoted as $P(A|B) = \frac{P(A \cap B)}{P(B)}$ where B denote the event which has already occured and A is the event for which we need to calculate the probability. $P(A \cap B)$ deote the chance of events A & B happening together, somethimes denoted as $P(A; B)$ or $P(A, B)$. This is known as the **joint probability**. Let $D1$ denote the items purcahsed from desk 1 and $D2$ denote the items purcahsed from desk 2. $P(D1 = 2|D1 + D2 \leq 5) = \frac{\frac{3}{36}}{\frac{10}{36}} = \frac{3}{10}$. Here the $D1 + D2 \leq 5$ event reduces the sample space size to 10 from 36.

In a retail store example, a customer may like one item which may not be liked by another customer. So the event of one customer liking an item is independent of another customer liking that item. So these two events are **independent events**. This is represented as $P(A|B) = P(A)$, $P(B|A) = P(B)$ & $P(A \cap B) = P(A) * P(B)$. Two events are **mutually exclusive events** if both events will never occur together. eg: Customer purchasing an item event and not purchasing an item events. This is represented as $P(A \cap B) = 0$ & $P(A|B) = P(B|A) = 0$.

3.1 Bayes Theorem

In many real world scenarios we don't know the joint probabiliites. Bayes theorem is a way to estimate **conditional probability** $P(A|B)$ without knowing the joint probability. Its computed using the individual probabilites of events (**prior probability**$(P(A))$, evidence $P(B)$) and **likelihood** $P(B|A)$, The proof of this theorem can be done as follows.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A,B)}{P(B)} = \frac{P(B,A)}{P(B)}$$
$$P(B|A) = \frac{P(B,A)}{P(A)} => P(B,A) = P(A) * P(B|A)$$

This expression we can substitute in previous one and get Bayes theorem proved. So the final formula for $P(A|B)$ can be written as Eq. 3.1.

$$P(A|B) = \frac{P(A) * P(B|A)}{P(B)} \qquad \text{where } P(B) > 0 \qquad (3.1)$$

3.2 Probability Distributions

We have around $8B$ world population as on 2022, but not all continents have equal population. Asia has around 60% of population and Antartica has close to 0%. So we say the population is distributed around the globe. Fig. 3.2 shows this distribution.

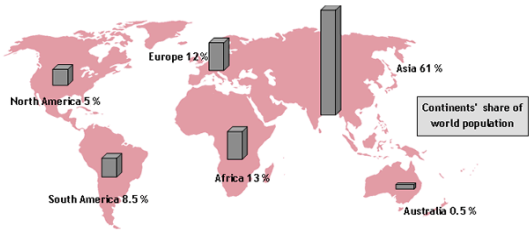


Figure 3.2: Population Distribution

If we choose one person randomly from the population, then what is the probability that he is an Asian? its around 0.60. Likewise if we assign probability to each possible inputs (in this example continents), we get a distribution and that is called **probability distributions**. When we plot random variable against its frequency we get **Probability distribution function** (PDF). Depending on the type of random variable we have two types of probability distributions **Discrete Probability Distribution** for which the PDF is called **Probability Mass Function** and **Continuous probability distribution** for which the PDF is called **Probability Density Function**. Any probability distribution must poses two properties,

1. $0 \leq P(x) \leq 1$ for any $x \in X$
2. The probability value for all possible value of X should sum up to 1
    - $\sum_{x \in X} P(x) = 1$ (X is discrete) & $\int_{-\infty}^{\infty} P(x)dx = 1$ (X is continuous)

- Any probability distribution either continuous or discrete, summarizes all information about about a random variable using real numbers called **parameters** (eg: μ, σ).
- In contrast to a histogram & barchart which are build from a sample, probability distribution is always build for population.

Suppose we have the height of each person (continuous random variable) recorded somewhere. What percentage of the population has a height of 150 cms and below? To answer such questions we can use **cumulative distribution function (CDF)** $P_x(X)$, that estimates the cumulative probability of a random variable at a point $X = x_t$. i.e It is the total probability distribution function up to a certain point $x_t \in X$. CDF can be derived from its probability distribution function as given in Eq. 3.2,

$$P_{x_t}(X) = \sum_{x \in X, x \leq x_t} P(x) or \int_{-\infty}^{x_t} P(x)dx \tag{3.2}$$

CDF plot is monotonically increasing function bounded in the interval $[0, 1]$. We can also find the probability of X above a particular threshold $(1 - P_x(X))$. In the following sections, we will discuss some of the most commonly used distributions in ML, its PDF, CDF and properties.

To understand the application of PDF and CDF, lets consider this example, suppose we want to find the percentage of people whose height is in between $170 - 180$ cms. We know that area has a population of $5M$. Its practically difficult to measure the height of everyone in that area. So we take a sample of reasonable size and get their measurements.

- Using the sample data estimate the parameters like μ, σ.
- Identify the type of distribution using QQ-plot type of testing . Let's say the height follows normal distribution.
- Construct the PDF & CDF curve for the distribution and we have the equation for the distributions.
- From the CDF identify the probability for end points 170&180.
- Calculate the absolute difference between the probability values. so we get the percentage value for people falling in this range
- Compute the actual count of people by using the percentage value computed in previous step and the total population count.

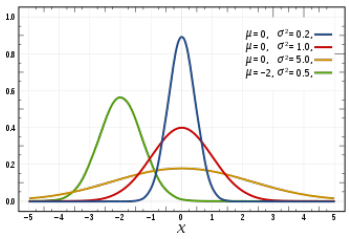
Lets consider another example of rain fall data. Suppose we have just 200 records of rain falls of past 30 years and we want to get the probability of ranin fall $X > 20$ cms. The sample we may have contains no records

with rainfall above 20 cms - so the probability so estimated would be 0.0. So percentile based method fails due to insufficient data. A better method would be fit a probability distribution to the data and using the CDF we can evaluate $P(x >= 20)$. Probability distribution is an alternative for the cases where non-parmametric methods like percentile values fails.

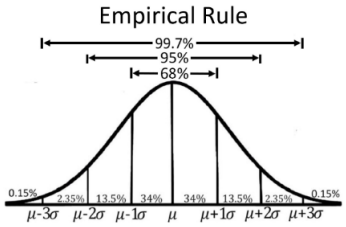
Another important concept to understand is the **Likelihood** which is often confused with probability. Likelihood estimates the best distribution by identifying the parameters (like μ, σ etc.) that would have generated a given data point. For eg, the height of person varies from continent to continent. So if we take a random person and found the height of that person, what is the likelihood that the person belong to Asia, Europe etc. Here Asia, Europe represents different distribtion (different parameters) and likelihood determines the best distribution that matches the given height.

3.2.1 Normal or Gaussian Distribution

We say the random variable X follows a normal distibution with mean μ and variance σ^2 , denoted as $X \sim N(\mu, \sigma^2)$. Univariate Gaussian distribution is a bell shaped curve as shown in Fig. 3.3 and the distribution function is given in Eq. 3.3. $X \sim N(0, 1)$ denote the **standard normal distribution**.



(a) Normal Curve



(b) 65-97-99

Figure 3.3: Gaussian or Normal Distribution

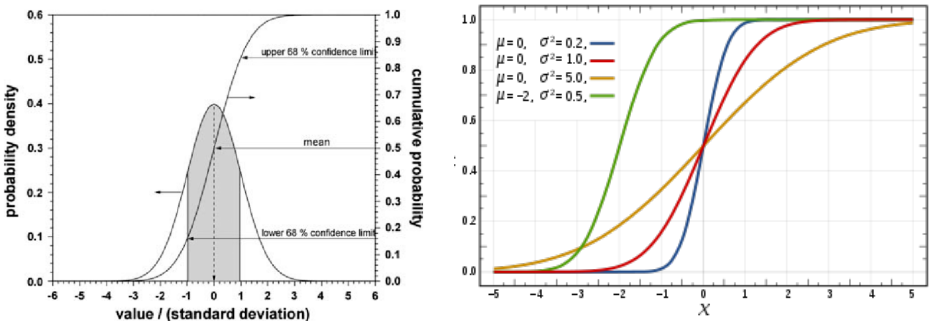
$$f(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\} \tag{3.3}$$

- The peak value is located at the mean μ and is symmetric about the line $x = \mu$ and CDF value will be 0.5 at $x = \mu$
- The value of probability decreases exponentially as we move away from the mean μ
- 68% - 95% - 99.7% of data points are within 1σ, 2σ & 3σ respectively from μ

Some real world cases where we observe normal distribution are birth weight of newborn babies, height of humans & diastolic blood pressure for men. The mean, median, variance and mode for gaussian distribution function are shown in Table 3.1. The cummulative distribtion of normal distibution along with its PDF is given in Fig. 3.4a and how change in μ, σ effects the CDF curve is shown in Fig. 3.4b.

Table 3.1: Statistics of Normal Distribution

Statistics	Function
Mean, Median, Mode	μ
Variance	σ^2



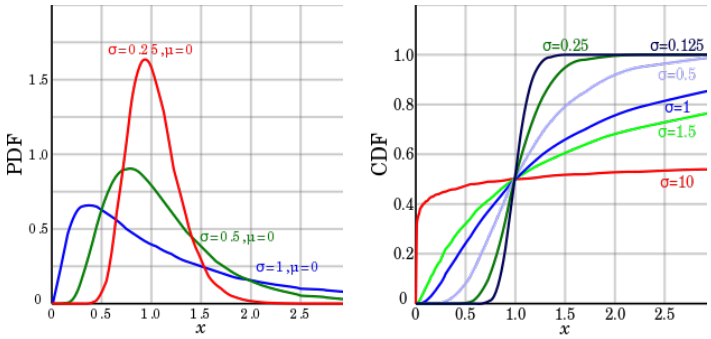
(a) CDF of Normal Distribution (b) μ & σ effect on CDF

Figure 3.4: PDF & CDF of Normal Distribution

3.2.2 Log Normal Distribution

A random variable X is said to follow to **heavy tailed distribution** or **log normal distribution**, if the natural logarithm of X follows normal distribution. We see lot of real world examples for this distribution. The length of some comments posted in internet forums are longer but most of the comments are shorter. Most users spent less time in reading blog or articles but some of them spend lot of time. Most of social media multimedia files are shorter in size but some of them are of high size.

As soon as we know X follows log normal distribution, we can create a new random variable Y by applying natural logarithm on X i.e $\ln(X) = Y =\sim N(\mu, \sigma^2)$ and treat Y as normally distributed. The PDF & CDF of log normal distribution is shown in Fig. 3.5a & Fig. 3.5b respectively.



(a) PDF (b) CDF

Figure 3.5: PDF & CDF of Log Normal Distribution

As the σ increases, PDF tends to have fatter tail. The statistics of Log Normal distribution are shown in Table 3.2.

Table 3.2: Statistics of Log Normal Distribution

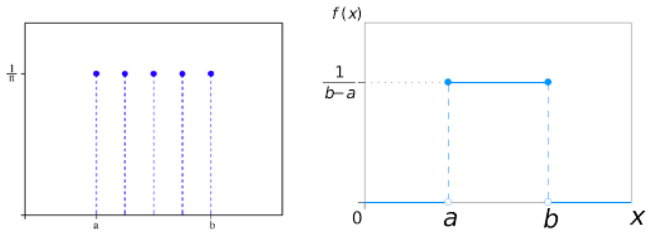
Statistics	Function
Mean	$\exp\left\{(\mu + \frac{\sigma^2}{2})\right\}$
Variance	$[\exp\{(\sigma^2 - 1)\} - \exp\{(2\mu + \sigma^2)\}]$
Median	$\exp\{(\mu)\}$
Mode	$\exp\{(\mu - \sigma^2)\}$

3.2.3 Uniform Distribution

A radnom variable X is said to follow uniform distribution when every possible value of X is equally likely. This distribution has two parameres a, b which represents the lower bound and upper bound of X. Dependng on X, there are two types of uniform distributions as shown in Fig. 3.6.

1. **Discrete uniform distribution** : X is discrete eg: Lucky draw, Rolling Dice
2. **Continuous uniform distribution** : X is continous eg: Random number generation in Python.

In general, when a point is randomly chosen in an interval (length), an area, or from a volume, we have a continuous uniform distribution and the



(a) Discrete Distribution (b) Continuous Distribution

Figure 3.6: Uniform Distribution PDFs

probability of an event is obtained by dividing the favorable length, area, or volume by the total length, area, or volume. The statistics for uniform distribution is given in Table **??**.

Uniform Distrbution Statistics		
Statistic	**Discrete Distribution**	**Continuous Distribution**
Mean, Median	$\frac{a+b}{2}$	$\frac{a+b}{2}$
Variance	$\frac{(b-a+1)^2}{12}$	$\frac{(b-a)^2}{12}$
Mode	NA	NA

3.2.4 Bernouli & Binomial Distributions

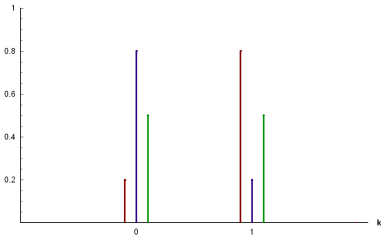
Bernouli distribution is a discrete distribution having only two outcomes (**success - 1** & **failiure -0**). This distribtion has only one parameter p $(0.0 \leq p \leq 1.0)$ that typically represents the probability of success. Probability mass function f of this Bernouli distribution over outcomes $k \in \{0, 1\}$ can be written as :

$$f(k, p) = p^k (1 - p)^{1-k} \tag{3.4}$$

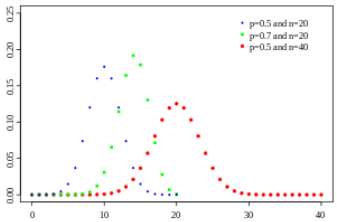
Binomial distribution is repeatation of Bernouli trails n times. This distribtion has one additional parameter n. $Bn(p, n)$. Suppose it is known that 2% of all credit card transactions in a certain region are fraudulent. If there are 50 transactions per day in a certain region, we can use a Binomial distribution calculator to find the probability for a certain number of fraudulent transactions out of n transactions. The formula for computing the probability of finding r fraudulent transactions out of n transactions is given below,

$$f(n, r, p) = \binom{n}{r} p^r (1 - p)^{n-r} \tag{3.5}$$

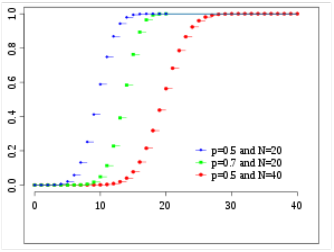
A binomial test which is based on binomial distribution could be conducted to make a decision on improving the security of existing online transactions. The PDF of Bernouli distribution, PDF and CDF of binomial distributions are shown in Fig. 3.7 and the statistics are given in Table **??**.



(a) Bernouli PDF



(b) Binomial PDF



(c) Binomial CDF

Figure 3.7: Distributions Curves

Bernouli & Binomial Distribution Statistics		
Statistic	**Bernouli**	**Binomial**
Mean	p	np
Variance	$p(1-p)$	$np(1-p)$
Median	$\begin{cases} 0, & 1-p > p. \\ 0.5, & 1-p = p \\ 1, & 1-p < p \end{cases}$	$\lfloor np \rfloor or \lceil np \rceil$
Mode	$\begin{cases} 0, & 1-p > p. \\ 0,1, & 1-p = p \\ 1, & 1-p < p \end{cases}$	$\lfloor (n+1)p \rfloor or \lceil (n+1)p \rceil - 1$

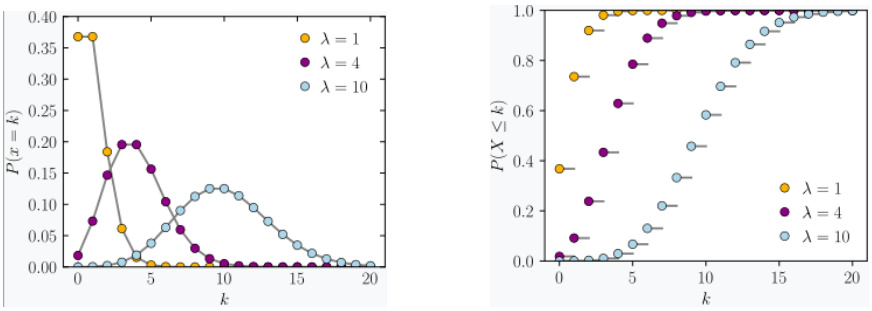
3.2.5 Poisson Distributions

Poisson distribution is a discrete probability distribution that can be used to predict the number of occurences of independent events in a given time period. This distribution is observed in many real world cases, for instance, a call center receives an average of 180 calls per hour, 24 hours a day. The

calls are independent, receiving one does not change the probability of when the next one will arrive. The number of calls received during any minute has a Poisson probability distribution with mean 3 (180 = 3 x 60). It is also used by insurance companies to conduct risk analysis (eg. predict the number of car accidents within a predefined time span) to decide car insurance pricing. This distribution has only one parameter λ that represents the mean rate (eg: number of calls per minute) of occurence of event.

The distribution function is given in Eq. 3.6 and the probability PDF & CDF are shown in Fig. 3.8.

$$P(x) = \frac{e^{-\lambda}\lambda^x}{x!} \tag{3.6}$$



(a) PDF (b) CDF

Figure 3.8: Poisson Distribution

3.3 Central Limit Theorem

Sampling distribution and central limit theorem are fundameanl concepts in probability and statistics. When we do analysis on the data, we don't know from which distribution the data comes from. Central limit theorem says *who cares it?*.

Let a random variable X follows **any distribution** with mean μ and standard deviation σ. From the population given, lets pick K samples each of size N independently. Let the samples denoted as K sets $[S_1, S_2, \ldots, S_K]$ and the corresponding means of each samples be $[\bar{X_1}, \bar{X_2}, \ldots \bar{X_K}]$ respectively.

Cental limit theorem says that, the means $[\bar{X_1}, \bar{X_2}, ..\bar{X_K}]$ follows normal distribution $N(\mu, \frac{\sigma^2}{N})$ as $N \to \infty$. This distribution is called **sampling distribution of sample means**. In real world scenarios, (as a rule of thumb) this holds good for $N \geq 30$. Using central limit theorem we can estimate the mean and variance of population which is not available to us.

As the mean value of samples follows normal distribution, we can use that to estimate confidence intervals, do t-test, Anova test or any other test that uses sample mean.

3.4 Confidence Interval

Developing good predictive models hinges upon accurate performance evaluation. Evaluating models on just one dataset could be misleading due to the fact that we are deciding the performance of model based on **point estimate** of parameters such as accuracy. When we take a different sample for testset the accuracy estimate is very likely to be different from the previous point estimate. So we need a measure to report uncertainity. Standard deviation is one simple way to report this uncertainity. **Confidence interval** is another method to set lower and upper bounds for the uncertainty. The true population parameter lies somewhere between the lower and upper bounds with some confidence level(say 95%) and this method is called **interval estimate**.

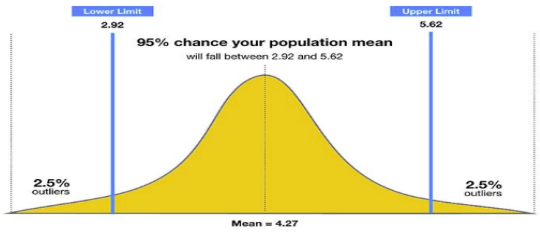


Figure 3.9: Confidence Interval

The **normal approximation interval** maybe the easiest and most classic way of creating confidence intervals. Using this method, we compute the confidence interval from a single training-test split. This is particularly attractive in deep learning where training step is expensive. This method is based on $z-$value & **standard error**. Let ACC_{test} denote the test accuracy, then the confidence interval is estimated using Eq. 3.7 where N denote sample size. z-value corresponding to 95 percentile could be substituted to estimate 95% confidence interval.

$$CI = ACC_{test} \pm z - \text{value} * \underbrace{\sqrt{\frac{1}{N} ACC_{test}(1 - ACC_{test})}}_{\text{Standard Error}} \tag{3.7}$$

Another way to estimate confidence interval is to use **bootstrapping method**. Use bootstraps (minimum 200 sample is recommended) to evaluate the metric for different samples. Then compute the 2.5 percentile and 97.5 percentile of metrics saved to get lower and upper bound for confidence

interval. Bootstrapping is computationally expensive than the normal approximation interval method, so it is commonly used in relatively small data sets.

High confidence level or small sample size is related to increase in the confidence interval width. We can say that the difference of two measurements (eg: accuracies) is statistically significant if confidence intervals do not overlap. However we cannot say that results are not statistically significant if confidence intervals overlap.

Let us assume we have a random variable X which follows any distribution with mean μ and standard deviation σ.

1. Case 1: We know the value of σ (Somebody has given us the population variance). From sample we can compute the mean $\bar{x}$. According to central limit theorem $\bar{x} \sim \text{Normal}(\mu, \frac{\sigma^2}{N})$. So the 95% confidence interval for μ is $[\bar{x} - 2\sigma, \bar{x} + 2\sigma]$.

2. Case 2: σ is unknown. Here we can use student t-distribution with $N - 1$ degrees of freedom to estimate the CI.

3.5 Hypothesis Testing

In our daily life we come across decision making situations like,

- Does men earn more salary than women? (Yes or No)
- New ML model perormance would be better than old model in production ? (Yes or No)

Hypothesis testing is a way to formalize these type of decision making where we have to choose one hypothesis between two conflicting hypothesis. Usually the hypothesis is coded as the parameter of some distribution (eg: mean salary of men $\mu = 20K$). In general, for any distribution, we do not know the true value of population parameters - they must be estimated. However, we do have hypotheses about what the true parameter values are. There are two types of hypothesis in decision making,

- **Null hypothesis (H_0)** : This is the hypothesis that needs to be tested. The null hypothesis is assumed to be true unless there is strong evidence against it - this is similar to how a person is assumed to be innocent until proven guilty by court.
- **Alternate hypothesis (H_1)** : This is the hypothesis we wish to accept. We, in particular, want to know whether the alternative hypothesis is correct.

Either of these hypothesis should contain the population parameter and in general, it is most convenient to always have H_0 contains an equals sign on parameter. The null and alternative hypothesis should be stated before any statistical test of significance is conducted. In other words, you technically are not supposed to do the data analysis first and then decide on the hypotheses afterwards.

A statistical test in which the alternative hypothesis specifies that the population parameter lies entirely above or below the value specified in H_0 is a **one-sided test** (or **one-tailed test**) and the test in which parameter lies in either side of the value specified in H_0 is a **two-sided test** (or **two-tailed test**). This is shown in the below example where μ denote average salary

$$
\begin{aligned}
H_0 &: \mu = 20K \\
H_1 &: \mu > 20K (\text{one sided}) \\
H_1 &: \mu! = 20K (\text{two sided})
\end{aligned}
$$

Whether you use a one-sided or two-sided test depends on the nature of the problem. Usually we use a 2-sided test. A one-sided test typically requires a little more theory. For example, suppose the null hypothesis is that the salaries of men and women are equal. A two-sided alternative would simply state that the salaries are not equal (salary of men can be more or less than women salary) where as the one sided test state that men make more money than women. Later is a strong statement as this tells us the direction of the difference in salary.

How do we choose between H_0 and H_1 ? This is done using the probability theory, we try to determine whether there is sufficient evidence against H_0. We reject H_0 only when the chance is small chance that H_0 is true. Since our decisions are based on probability rather than certainty, we can make two types errors,

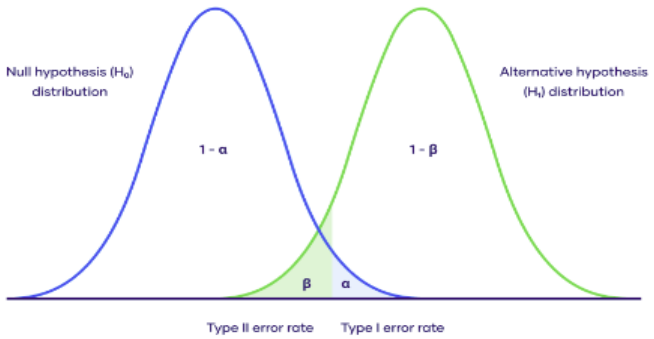


Figure 3.10: Type I and Type II Errors

- **Type I error** : We reject the null hypothesis when the null hypothesis is true. The probability of Type I error is denoted as α (typical values are 0.05 or 0.01), where

$$\alpha = P(\text{Rejecting } H_0 | H_0 \text{ is True}) \tag{3.8}$$

- **Type II error** : we accept the null hypothesis when it is false. The probability of Type II error is denoted as β, where

$$\beta = P(\text{Accepting } H_0 | H_0 \text{ is False}) \tag{3.9}$$

Its not possible to find the best α, β values rather there is a trade off - if we try to decrease any of these the other will increase. As we increase sample size, both will reduce due to the reduction in sampling error.

Now lets look into a problem that can be answered using hypothesis testing. Suppose we have two cities C_1 & C_2 and want to know is there any significant difference between mean heights of people in two cities. We need to be absolutely sure about it and need to quantify this. Let μ_1, μ_2 be the mean of population heights of cities C_1 & C_2 respectively. Hypothesis testing involves the following four steps,

1. Design the experiment
2. Design a test statistic (X)
3. Design Null Hypothesis (H_0) and Alternate Hypothesis (H_1). We should choose the H_0 such that p-value computation is easy and feasible.
4. Compute p-value and accept or reject H_0. Compute probability value p assuming H_0 is true i.e $p = P(X|H_0)$. Reject H_0 (or accept H_1) if p value is very small (often $< 5\%$), otherwise accept it (H_1 is rejected).

Design the experiment: It is almost impossible to measure the heights of every individual in the cities as the population size of cities are typically in millions. How do we design our test statistics for this problem ? We can take a sample of reasonable size from both cities and measure individual heights.

Test statistic: We can consider **sample means** for designing our test statistics. Test statistics can be designed as the difference in mean heights of samples ($\mu_1 - \mu_2$).

The null & alternate hypothesis can be designed as follows :

- H_0 : There is no considerable difference between mean heights of population of both cities.

- H_1 : There is considerable difference between mean heights of population of both cities.

How to compute p-value ? p-value is computed using **permutation method**. In permutation test we take two samples from both the cities randomly. Let S_1 & S_2 be the random samples from cities C_1 & C_2 respectively. Let Δ be the absolute difference between mean heights of two samples S_1 & S_2. Δ is our observation and cannot be wrong. The basic steps of permutation test are :

1. Mix both the sets S_1 & S_2 and jumble it to form a new set S . As soon as we mix S_1 & S_2 and jumble it to create S, we assume that, there is no difference in mean heights (We don't have two cities rather we have only one city C). Here we make an assumption that H_0 is true.

2. Randomly sample two disjoint subsets $S_1 \& S_2$ from S. This step is known as **resampling** step.

3. Repeat resampling step some large number of times.

    - Compute the mean heights of set S_1 & S_2 in each of the resampling step i and compute the difference between means as δ_i.

4. Sort the δ_i is in ascending order. In the sorted list, check the position of Δ, and compute the probability (p-value). $p-value = \text{Prob}[\delta_i > \Delta]$

5. Reject or Accept H_0 based on the p-value at specific significance level α (reject H_0 if p-value $< \alpha$, Typically We select α less than 0.05, in medical fields value 0.01 is used).

3.6 Power Transformations

Machine learning algorithms like Linear Regression and Gaussian Naive Bayes assume the numerical variables have a Gaussian probability distribution. Your data may not have a Gaussian distribution and instead may have a Gaussian-like distribution or distribution with skews. As such, you may be able to achieve better performance (or converge faster) on a wide range of machine learning algorithms by transforming input and/or output variables to have a Gaussian or more-Gaussian distribution.

- A power transform will make the probability distribution of a variable more Gaussian. This is often described as removing a skew in the distribution, although more generally is described as stabilizing the variance of the distribution.

- The transformation is done using equations for positive, negative values of variable and λ a hyper prarameter which is estimated using maximum likelihood method. Popular power transformation methods are Box-Cox & Yeo-Johnson method.

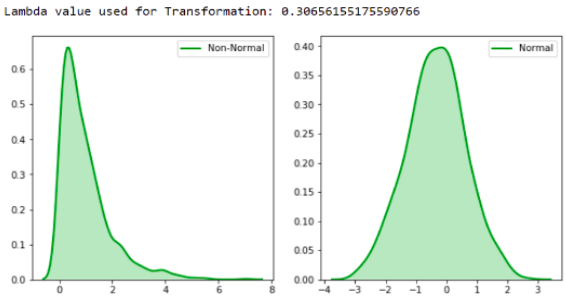


Figure 3.11: Power Transformations

3.7 Proportional Sampling

There are some situations where we need to sample the data points based on the value of the data points , i.e the higher values have higher chances of being selected. **Proportional sampling** or **Stratified sampling** is a method to do this type of sampling. Suppose our dataset D contains N samples $[x_1, x_2, \ldots, x_N]$. The steps to do proportional sampling are as follows.

- Sort the numbers of D in ascending order. Let D' be the sorted dataset.
- Normalize the data points of D' to fall in the range $0 - 1$.
- Find the cumulative sum of normalized data points. Let the cummulative sum be $[c_1, c_2, \ldots, c_N]$.
- Generate a random number r uniformly in the range $[0.0 - 1.0]$.
- Use the index j where $c_j < r \leq c_{j+1}$ and pick the j^{th} element from D' as sampled element.

4

Know Your Data

We don't know anything about the data when we start exploring it.**Exploratory Data Analysis** is a task of analysing data using linear algrebara, statistics and visualization tools etc. at early stages of model developement.

- We call each entry in a dataset as **datapoint**, **vector** or **observation**. Each dimension of vector is called an **independent variable**.
- The class label is called **response label**, **output variable** or **dependent varaible**.

Doing basic EDA, regarding each attribute makes it easier to:

- Fill in missing values
- Spot outliers
- Smooth noisy values in the column

4.1 Data Objects and Attribute Types

Data sets are made up of **data objects** which represents an entity, for eg: patients records in a medical database, students, professors, and courses records in university database.

- Data objects are typically described by its **attributes**
- Different terminologies are used for attibutes in different fields
  - data warehousing => **Dimension**
  - machine learning => **Feature**
  - statistics & mathematics => **Variable** or **Random Variable**
  - database => **Attribute**

  - non-technical folks => **Column**

- **Univariate distribution** & **Multivariate distribution**

  - data distribution involves only one attibure vs more than one

Depending on the set of possible values that it can take, an attribute can be of three types namely Nominal, Ordinal and Numeric.

Nominal	**Ordinal**	**Numeric**
"relating to names" or labels or a category	Like nominal but a specific order can be assigned for each value	A measurable qunatity - ie Quantitative
Arithmetic operation is invalid	Arithmetric Opearion is invalid	Aritmetic operation is valid
Only mode is defined	Only median is defined	μ, σ, Mode, Median are defined
eg: Zip codes, Country names, Roll number	eg: Drink size(small, medium, large), Professional rank(Engineer, Senior Engineer), Grade	Income, Weight, Height
Encoding is required for model to process the data	Numbers can be readilly substituted	It can be directly processed
One hot encoding eg: 7 dimension encoding for VIBGYOR - but doesn't scale well for large dimensions like zip codes Target label based encoding - Mean substitution or probability distribution encoding for regression and classification problems respectively	Number substitution works but best encoding may require some domain knowledge & trail and error	No need of encoding but scaling can be done
Symmetric binary (Gender M/F) & Assymetric binary (Converter/Non-converter) types, Continuous vs Discrete types	-	Ratio scaled (Length, Weight - ratio holds good if we switch from one unit to other), Interval scaled - (Temperature - Ratio doesnt hold),

4.2 Handling missing values by imputation

The data may be missing in dataset due to corruption or missed it at the time of data aquisition and this can be handled by imputation methods.

1. Mean, Median, Mode imputation - group feature values by its class label (for classification problems) and use mean, mode, median as applicable
2. Missing value as source of information - Treat missing value as another category of feature value to consider
3. For regression problems, use the feature column and consider Mean, Median, Mode as applicable to the type of attribute

4.3 Impact of Scale and Standardization

Suppose we have 2D data having two features f_1, f_2 and have the range of values $0-100$, $0-1$ respectively. Let $p_1 = [23, 0.2], p_2 = [28, 0.2], p_3 = [28, 1.0]$ be three datapoints. When we try to compute the Euclidean distance between pair of points, we get

$$d(p_1, p_2) = 5.0$$
$$d(p_2, p_3) = 0.64$$

Here $d(p_1, p_2)$ appears to be too large than $d(p_2, p_3)$ and leads to conclusion that p_2, p_3 are very close. But logically if we think there is 80% difference in f_2 values. This shows the Euclidean distance is sensitive to scale of the data.

- Some algotihms like KNN, K-means which uses Euclidean distance will end up with wrong results if we do not standardize the data.
- Algorithms that uses dot product such as Linear Regression, Logistic Regression, SVM are also feature scale sensitive.
- Algorithms that are based on decision trees are independent of scale of data. These algorithms uses comparison on featue values rather than distance computation or sum of products. So its independent.

4.4 Measuring the Central Tendency of Data

Given an attribute, where do most of its values fall ? Measures of **central tendency**, which measure the location of the middle or center of a data distribution. Popular measures used in central tendancy are

- Mean (average value)
- Meadian (middle value)
- Mode (most common value)
- Mid Range

Suppose that we have some attribute X, like salary, which has been recorded for a set of objects. Let $x_1, x_2, \ldots, x_N$ be the set of N observed values or observations for X. If we were to plot the observations for salary, where would most of the values fall?

The most common and effective numeric measure of the "center" of a set of data is the (arithmetic) **mean** which is computed using Eq. 4.1.

$$\overline{x} = \frac{\sum_{i=1}^{N} x_i}{N} \tag{4.1}$$

Sometimes, each value x_i in a set may be associated with a weight w_i for $i = 1, 2, \ldots, N$. The weights reflect the significance, importance, or occurrence frequency attached to their respective values. The **weighted arithmetic mean** or the **weighted average** can be computed using Eq. 4.2

$$\overline{x} = \frac{\sum_{i=1}^{N} w_i x_i}{N} \tag{4.2}$$

- A major problem with the mean is its sensitivity to extreme values (outlier). Even a small number of extreme values can corrupt the mean. eg: Income to measure and the sample contains richest person's income.
- To offset the effect caused by a small number of extreme values, a **trimmed mean** is recommended. Its the mean obtained after chopping off values at the high and low extremes. For example, we can sort the values observed for salary and remove the top and bottom 2% before computing the mean.

For skewed (asymmetric) data, a better measure of the center of data is the **median**.

- Median is the middle value in a set of ordered data values.
- It is the value that separates the higher half of a data set from the lower half.

- If number of examples N is odd, then the median is the middle value of the ordered set.
- If N is even, then the median is not unique.
  - If X is a numeric attribute in this case, by convention, the median is taken as the average of the two middlemost values.

The **mode** is another measure of central tendency. The mode for a set of data is the value that occurs most frequently in the set.

- It can be determined for numeric and nominal attributes.
- It is possible for the greatest frequency to correspond to several different values, which results in more than one mode.
- Data sets with one, two, or three modes are respectively called **unimodal**, **bimodal**, and **trimodal**. In general, a data set with two or more modes is **multimodal**.
- Mode doesnt exist if each data value occurs only once.

Midrange can also be used to assess the central tendency of a numeric data set. It is the average of the largest and smallest values in the set.

In a unimodal prefect symmetric distribution of data, the mean, meadian and mode all at the same centre as shown in Fig. 4.1(a).

- Data in most real world applications are not symmetric.
- Data may be either **positively skewed**, where the mode occures at a value that is smaller than the median.
- Data may be **negatiely skewed** where mode occures at a value greater than median.

4.5 Measuring the Dispersion of Data

How are the data spread out ? i.e we would like to have an idea of the **dispersion** of data. The most common data dispersion measures which are useful in spotting outliers are

- Range
- Quartiles
- Interquartile range

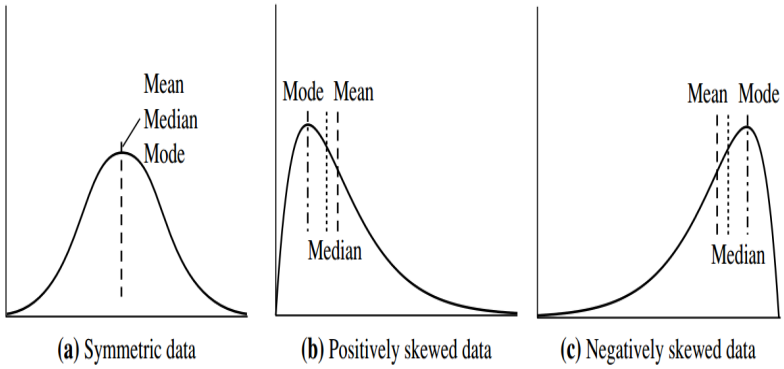


Figure 4.1: Symmetric versus positively and negatively skewed data

- Variance
- Standard Deviation
- Median Absolute Deviation

4.5.1 Range, Percentiles and Qunatiles

Let $x_1, x_2, x_3, \ldots, x_N$, be a set of observations for some numeric attribute X sorted in ascending order.

- The **Range** of X is the differrence between largest and smallest value of X.
- **Qunatile** indicate where does the data point lies in the sorted list.
- **Percentile** divides X into 100 parts each of 1 unit, where as a **Quartile** divides data points into 4 equal parts - namely Q_1, Q_2, Q_3, Q_4) represents $0-25\%$, $25-50\%$, $50-75\%$, $75-100\%$ parts respectively.
- Quantiles are useful in ecommerce applications. eg: suppose X_s represent the delivery time for an order, then percentiles like 98% indicates customer satisfaction - meaning 98% users received the product in time.
- **Inter Quartile Range** (IQR) is the difference between 75^{th} percentile and 25^{th} percentile ($Q_3 - Q_1$), it gives the range covered by the middle half of the data.

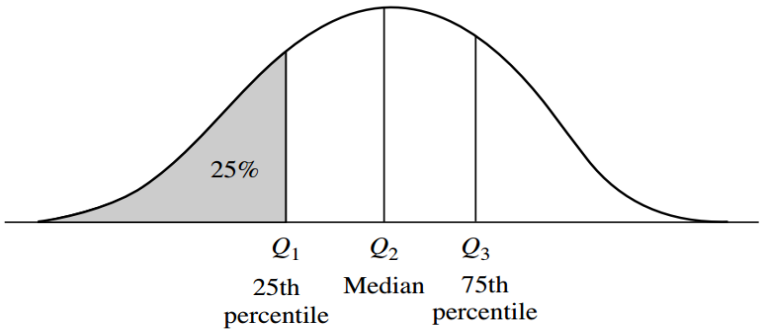


Figure 4.2: A plot of the data distribution for some attribute X

4.5.2 Variance and Standard Deviation

The **variance** and **standard deviation** indicate how *spread* out a data distribution is. These measures represnts how far most of the data points from the mean. Lower the variance/standard deviation closer is the point to the mean. The computation of these measures are scalable in large databases.

The variance (σ^2)of N observations $x_1, x_2, x_3, \ldots, x_N$ for an attribute X is given in Eq. 4.3, where $\bar{x}$ is the mean value defined in Eq. 4.1. Standard deviation (σ) is the square root of variance.

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 = \left(\frac{1}{N}\sum_{i=1}^{N}x_i^2\right) - \bar{x}^2 \tag{4.3}$$

- Chebyshev's inequality says, an observation is unlikely to be more than several standard deviations away from the mean. This idea can be used for detecting outliers.

- We can compute the standard deviation of means taken from many sample and such a standard deviation is called **Standard Error**

Median absolute deviation (MAD) is similar to standard deviation.

- Instead of using mean, in MAD we use median as a central tendency term.

- We take the absolute difference of each data point from the median and take the median of the absolute deviations.

- Since we are using the median as a cental tendancy term, it is more robust against outliers.

4.6 Co-Variances and Correlation

Imagine we have two random variables X, Y represtnts the heights & weights respectively of students. There are three ways to measure the relationship between X and Y.

1. Co-variance
2. Pearson correlation coefficeient
3. Spearman rank coefficient

How one variable varies with respect to the mean of other variable ? - its Covariance. Covariance between X, Y is measured as follows and looks similar to variance.

$$cov(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \tag{4.4}$$

$$cov(X,Y)\begin{cases} +ve & X,Y \text{changes in the same way roughly (if } X\uparrow Y\uparrow \text{, or if } X\downarrow Y\downarrow) \\ -ve & X,Y \text{changes in the opposite way roughly } (X\uparrow Y\downarrow \text{, or if } X\downarrow Y\uparrow) \\ 0 & X,Y \text{are independent} \end{cases}$$

- Stronger value of covariance indicates the strength of relationship between X and Y.
- Covariance has a drawback that it gives different values if we measure X, Y in different scale.eg: Cov (Height in feet, Weight in Kg) and Cov(Height in cms, Weight in pound) will be different.

4.6.1 Pearson correlation coefficient

Pearson correlation coefficnet ρ, is a slight modification of covariance and is computed uisng Eq. 4.5. This metric addresess the scaling problem associated with covariance and quantify the relationship between X and Y. Unlike covariance, ρ is bounded in $[-1, 1]$

Fig. 4.3 shows various correlation types and correlation values. This type of scatterplots gives a hint about any existing correlataion between variables and hence the *linear relationship* between paired data.

$$\rho = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} \tag{4.5}$$

We can categorise the type of correlation by considering as one variable increases what happens to the other variable:

- $0 < \rho <= +1$ **Positive correlation** - the other variable has a tendency to increase. $\rho = +1$ is perfect +ve correlation and looks like a line.
- $-1 <= \rho < 0$**Negative correlation** - the other variable has a tendency to decrease. $\rho = -1$ is perfect -ve correlation and looks like a line.
- $\rho = 0$ **No correlation** - the other variable does not tend to either increase or decrease.
- $\rho = +1 or -1$ is independent of the slope of line. But there may be quadratic or other relationship between variables

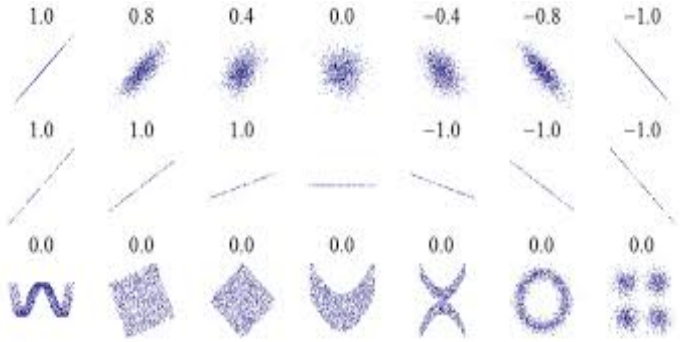


Figure 4.3: Pearson Correlation coefficeient

- Correlation coefficeient can tell only linear relatioship
- The existence of a strong correlation does not imply a causal link between the variables. i.e *Correlation doesnt imply Causation*
- We assume pair of values are bivariate normally distributed
  - In practice this assumption is checked by requiring both variables to be individually normally distributed.

4.6.2 Spearman's Correlation

To understand **Spearman's correlation** ρ_s , it is necessary to know what a **monotonic function** is. Fig. 4.4 shows the type of monotonic functions.

The calculation of and subsequent significance testing of it requires the following data assumptions to hold:

- Variables X, Y are monotonically related

Table 4.1: Level of Correlation

ρ	Level of Correlation
0.00-0.19	Very weak
0.20-0.39	Weak
0.40-0.59	Moderate
0.60-0.79	Strong
0.80-1.0	Very strong

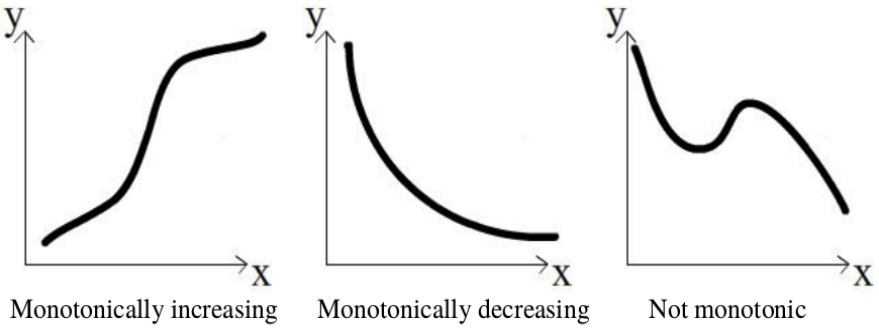


Figure 4.4: Monotonic function and its types

- Unlike Pearson's correlation, there is no requirement of normality and hence it is a nonparametric statistic.

Let us consider some examples to illustrate it. The following table gives X and Y values for the relationship $Y = e^X$. From the graph we can see that this is a perfectly increasing monotonic relationship. We get the coefficeient as 1, reflecting the perfect monotonic relationship.

- Spearman's correlation works by calculating Pearson's correlation on the ranked values of this data. Ranking (from low to high) is obtained by assigning a rank of 1 to the lowest value, 2 to the next lowest and so on.
  - First we need to create two random variables r_X, r_Y the ranks of X and Y respectively.
  - Instead of using X, Y as input to pearson correlataion coefficent, we can use r_X, r_Y and we get Spearman's correlation coefficeient.
- ρ_s 0 doesnt mean the values are not related. There may be a quadratic relationship between variables ($Y = X^2$).

Spearman's correlation coefficeient is computed using Eq. 4.6.

$$\rho_s = \frac{Cov(r_X, r_Y)}{\sigma_{r_X}\sigma_{r_Y}}. \tag{4.6}$$

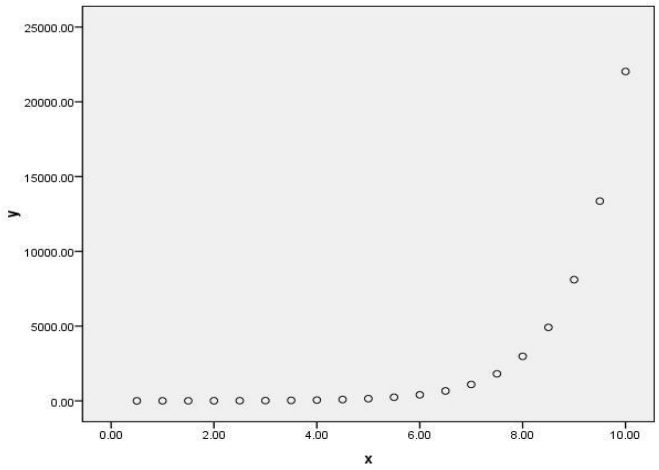


Figure 4.5: $Y = e^X$

- The advantage of Spearman Rank is that it can handle small number of outliers in data.
- If two random varibles X, Y are correlated well (high corelation coefficient), we cannot conclude that X causes Y or Y causes X.
- The non linear relatioship can be captured using **regression techniques**.

4.7 Measuring Data Similarity and Dissimilarity

This section presents **similarity** and **dissimilarity** measures, which are referred to as measures of proximity. A similarity measure for two objects, i and j, will typically return the value closet to 0 if the objects are unalike and value close1 if objects are identical. A dissimilarity measure works just the other way around.

Two data structures that are commonly used in the above types of applications are:

1. **Data matrix** (used to store the data objects)
2. **Dissimilarity matrix** (used to store dissimilarity values for pairs of objects)

4.7.1 Data Matrix versus Dissimilarity Matrix

Let the objects are $x_1 = x_{11}, x_{12}, x_{13}, \ldots x_{1d}$ and $x_2 = x_{21}, x_{22}, x_{23}, \ldots x_{2d}$, and so on. For brevity, we hereafter refer to object x_i as object i.

- **Data matrix**: This structure stores the N data objects in the form of a relational table, or N-by-d matrix (N objects $\times$ d attributes):

$$\begin{bmatrix} x_{11} & x_{12}\dots & x_{1f}\dots & x_{1d} \\ x_{21} & x_{22}\dots & x_{2f}\dots & x_{2d} \\ x_{31} & x_{32}\dots & x_{3f}\dots & x_{3d} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N1} & x_{N2}\dots & x_{Nf}\dots & x_{Nd} \end{bmatrix}$$

- **Dissimilarity matrix**: This structure stores a collection of proximities that are available for all pairs of N objects. It is often represented by an N-by-N table:

$$\begin{bmatrix} 0 & & & \\ d(2,1) & 0 & & \\ d(3,1) & d(3,2) & 0 & \\ \vdots & \vdots & \vdots & \vdots \\ d(N,1) & \dots & d(N,f)\dots & 0 \end{bmatrix}$$

  where $d(i,j)$ is the measured dissimilarity or "difference" between objects i and j.

Measures of similarity can often be expressed as a function of measures of dissimilarity as in Eq. 4.7. For example,

$$sim(i,j) = 1 - d(i,j) \tag{4.7}$$

- Many clustering and nearest-neighbor algorithms operate on a dissimilarity matrix.
- Data in the form of a data matrix can be transformed into a dissimilarity matrix before applying such algorithms.

4.7.2 Dissimilarity of Numeric Data: Minkowski Distance

In this section, we describe distance measures that are commonly used for computing the dissimilarity of objects described by numeric attributes. These measures includes

- **Manhattan**
- **Euclidean**
- **Minkowski distances**
- **Cosine Similarity**

- **Jaccard Similarity**
- **Tanimoto distance**

In most of the cases, the attributes are scaled before applying distance calculations. Let $i = (x_{11}, x_{12}, \ldots, x_{1d})$ and $j = (x_{21}, x_{22}, \ldots, x_{2d})$ be two vectors we need to calulate the distance between. A well-known measure is the Manhattan (or city block) distance, named so because it is the distance in blocks between any two points in a Manhattan city. It is defined in Eq. 4.8

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{id} - x_{jd}| \tag{4.8}$$

The Euclidean distance between objects i and j is defined in Eq. 4.9

$$d(i,j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{id} - x_{jd})^2} \tag{4.9}$$

Minkowski distance is a generalization of the Euclidean and Manhattan distances. It is defined in Eq. 4.10, where h is a real number such that $h \geq 1$.

$$d(i,j) = \sqrt[h]{|x_{i1} - x_{j1}|^h + |x_{i2} - x_{j2}|^h + \cdots + |x_{id} - x_{jd}|^h}, \tag{4.10}$$

Norm is the length of a vector. We have L_1 norm, L_2 norm, L_p norm, L_∞ norm. Then Minkowiski distance can be summarized as follows,

- $h = 1$ or L_1 norm => Manhattan distance
- $h = 2$ or L_2 norm => Euclidean distance
- $h = 2$ or L_∞ norm => Supremum distance / L_{max} / **Chebyshev distance** / **uniform norm**
  - To compute it, we find the attribute f that gives the maximum difference in values between the two objects and return the absolute value of the difference

  , and given in Eq. 4.11

- $$[label =]d(i,j) = \lim_{h \to \infty} (\sum_{f=1}^{p} |x_{if} - x_{jf}|^h)^{\frac{1}{h}} = \max_f |x_{if} - x_{jf}| \tag{4.11}$$

Fig. 4.6 shows an example of these distance metric. It is also posssible to assign different weights to different dimensions for these distance measures.

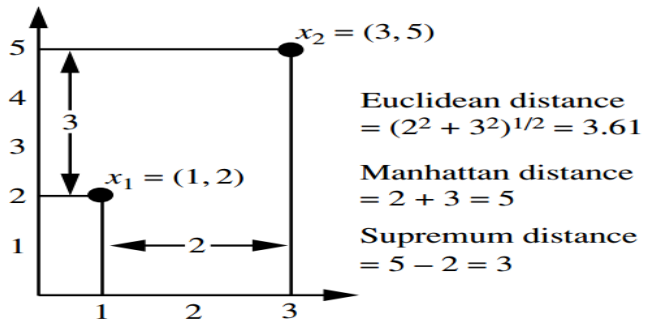


Figure 4.6: Euclidean, Manhattan, and Supremum distances

4.7.3 Cosine Similarity

One popular metric to compare two text documents or give a ranking of documents with respect to a given vector of query word is the cosine similarity. A document can be represented by thousands of attributes, each recording the frequency of a particular word (such as a keyword) or phrase in the document. Thus, each document is an object represented by what is called a term-frequency vector.

- Term-frequency vectors are typically very long and sparse. Some applications using such structures include information retrieval and text document clustering.
- The traditional distance measures that we have studied in this chapter do not work well for such sparse data.
- For example, two term-frequency vectors may have many 0 values in common, meaning that the corresponding documents do not share many words, but this does not make them similar.
- We need a measure that will focus on the words that the two documents do have in common, and the occurrence frequency of such words. Cosine similarity does exactly the same.

Let x and y be two vectors created for two documents for comparison. Using the cosine similarity function in Eq. 4.12, where $\|x\|$ is the Eucleadian norm of the vector $x = (x_1, x_2, \ldots, x_d)$, defined as $\sqrt{x_1^2 + x_2^2 + \cdots + x_d^2}$. Conceptually it is the length of the vector.

$$sim(x, y) = \frac{x.y}{\|x\|\|y\|} \tag{4.12}$$

- This computes the cosine of the angle between vectors x and y.

- A cosine value of 0 means that the two vectors are at 90 degrees to each other (orthogonal) and have no match.

When attributes are binary-valued, we can compute **Tanimoto distance** between the document vectors given in Eq.4.13. This measure is frequently used in information retrieval.

- Then $x.y$ is the number of attributes possessed (i.e., shared) by both x and y.
- $x.x$ is the number of words present in document vector x.
- $y.y$ is the number of words present in document vector y.
- It is the ratio of the number of attributes shared by x and y to the number of attributes possessed by x or y. i.e Intersection over Union for bianry vectors.

$$sim(x, y) = \frac{x.y}{x.x + y.y - x.y} \tag{4.13}$$

5

Know Your Features

In this chapter we introduce some techniques to extract and select features.

5.1 Feature Extraction and Feature Selection

The raw text data is a sequence of symbols/words which we cannot feed directly to the algorithms. We need to extract features from text and optionally select features before feed into models. Feature extraction and Feature selection differes fundamentally.

- **Feature extraction** consists in transforming arbitrary data, such as text or images, into numerical features usable for machine learning.
- **Feature selection** is a machine learning technique applied on extracted numerical features. Not all features are equally important and feature selection helps to pick the most important features.

The most common ways to extract numerical features from text content, are:

- **Tokenizing :** We can tokenize strings and give an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.
- **Counting :** Count the occurrences of tokens in each document.

We call **vectorization** the general process of turning a collection of text documents into numerical feature vectors. As most documents will typically use a very small subset of the words used in the corpus, the resulting matrix will have many feature values that are zeros (sparse-matrix).

5.2 Text Feature Extraction

One of the simple technique to convert text to vector is **Bag of words**. Let d is the number of unique words considering all reviews. d can be very large, but in review not all words will be present so the vector representation will be sparse. In BoW, we are not considering the sematntic meaning of words. Eg (tasty , delicious) , (affordable, cheap)

- Each review can be considered as one **document** and collection of all documents is called **corpus**.
- When converting text to vector, similar text should result in similar vectors.
- The problem with this method is that, if the english meaning of documents are completely different, there may be chance that the differene of corresponding vectors may be less due to common words.
- A variant of BOW, is the **binary bag of words**, instead of count it represents whether the word is present or not. Here the difference indicate the number of words that differ.
- **Stop words :** These are the words which are not important. Negative words like 'not' should be considered differently because it changes the meaning. Stop words are removed as as preprocessing step in NLP.

The basic operations in NLP are:

- Tokenization
- Removal of stop words
- Converting words to lower case letters
- **Stemming :** Find root word or base word . Eg. Walking $\rightarrow$ walk.
- **Lemmetizzation :** Breaking up a sentence into words. Breaking is some time dependent on context. eg: New York we cant break into New and York

Suppose we have two food reviews r_1, r_2.

- r_1 : This pasta is very tasty and affordable
- r_2 : This pasta is not tasty and affordable

The two reviews have entirely different meaning, but after removal of stop words, the reviews r_1, r_2 are exactly the same thing. To avoid the probelm with **unigrams**, we can use **bigram** or **tri-gram** cab be used.

- Bigrams basically means a pair of words. Tri-gram consists of three words.
- Using unigram causes lossing the sequence of words. As we use bigram/n-gram the sequence of words is retained and the number of dimensions will also increase.

5.2.1 Term-frequencey (TF), inverse document frequency(IDF)

TF-IDF, BoW are all parts of information retrieval. Using search engines, we submit our query which consits of few words, but the serach results will have thousands of pages and the number of words are also high. TF-IDF & BoW are used by search engine to retrieve the most relavant word to the query submitted.

Suppose we have a corpus D with reviews (documents) $d_1, d_2, \ldots, d_N$.

- **Term Frequencey (TF) :** The TF of a word w_j in a document (d_i) is defined as the number of times the word w_j occured in d_i to the total number of words in d_i. This says how often w_j occures in a document. (Probability)

$$TF(w_j, d_i) = \frac{n(w_j)}{\sum_{k \in d_i} n(w_k)} \tag{5.1}$$

- TF gives higher weigtages to words that appears most frequently within a document
- **Inverse Document Frequency (IDF) :** IDF is always measured using corpus D. The IDF is the logarithm of ratio of total number of documents (D) to the number of documents (n_i) in which w_i is present.

$$IDF(w_i, D) = \ln\left(\frac{D}{n_i}\right) \tag{5.2}$$

- IDF gives lower weigtages to words that are present in almost all documents

To assign a weigtage to the document d_i, we take the product of TF and IDF.

$$TF - IDF = TF(w_j, d_i) * IDF(w_i, D) \tag{5.3}$$

- So taking the product of TF & IDF as a measure, we can select the document which is rare but the word is frequent in the document.

- In TF-IDF we are multiplying two terms, taking log helps to avoid giving much higher weightage to IDF term.
- TF-IDF doesn't consider the semantic meaning.

5.2.2 Word2Vec

As the name suggests **Word2Vec** (W2V)is an algorithm which takes a text (words) as input and convert each word to a vector. This is one of the state of the arts method developed by Google. W2V model is trained on Google news documents.

- Semantic meaning between words. eg : (tasty, delicious) is considered in W2V
- Learns relationship between words automatically. eg: (males, females), (king, queen), (Canada, Ottawa) etc.
- For converting words to vector, it considers the words in the neighbourhood, if the words are within a neighbourhood corresponding vectos are also similar.
- Unlike BoW, W2V is a dense d-dimesnional vector. d is typically $50, \ldots, 300$ dimensions etc.

There are two variants of word2vec, **average w2v**, **weighted tf-idf w2v**. Lets assume our document d_1 contains k words, $w_1, w_2, \ldots, w_k$.

$$\text{avg-w2v} = \frac{\text{w2v}(w_1) + \text{w2v}(w_2) + \cdots + \text{w2v}(w_k)}{k} \tag{5.4}$$

$$\text{TF-IDF w2v}(r_1) = \frac{\sum_{i:\text{words}} \text{tf-idf}_i * \text{w2v}(w_i)}{\sum_{\text{tf-idf}_i}} \tag{5.5}$$

5.3 Featurization and Feature Engineering

Feature engineering is careful processing of features to add new features that we think would help to improve model performance. Its the most important aspect of machine learning as good featurization usually leads to good performance of machine learning models and outcomes. As an example,

- Instead of using variables x, y, z you decide to use $\log x - \sqrt{y * z}$ instead, because your engineering knowledge tells you that this derived quantity is more meaningful to solve your problem. You get better results than without.

5.3.1 Fourier Decomposition

Fourier decomposition is a popular technique used to represent time series data. Fourier decomposition is widely used in physics, applied mathematics, signal processing etc.

- Given a composite wave, we can break it into multiple components each having different frequencies using the fourier transform.
- It transforms time domain to frequency domain. There are cases where it is important to represent the singnal in frequency domain than the time domain. For eg: ecommerce sales data, where we are interested in frequency of purchase of products.
- Represnting the signal as a vector of frequency and amplitude is called **fourier representation** of data.

5.3.2 SIFT - Features

Scale Invariant Featue Transformation (SIFT) is a popular feature extraction method for images.

- An image can be represented as a set of key points which are represented as vectors.
- If the image is rotated or rescaled its SIFT features wont vary much.
- Typically SIFT feature is represnted as 128 dimensions.
- SIFT can robustly identify objects even among clutter and under partial occlusion, because the SIFT feature descriptor is invariant to uniform scaling, orientation, illumination changes, and partially invariant to affine distortion.
- Applications include object recognition, robotic mapping and navigation, image stitching, 3D modeling, gesture recognition, video tracking, individual identification of wildlife and match moving.
- SIFT keypoints of objects are first extracted from a set of reference images and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors.

5.3.3 Moving Window for Time Series Data

Moving window is a another technique used for feature engineering in time series domain. Eg. ECG data. Moving widow holds only the time series data in a specific interval. There are many ways of extracting useful features from time series data. A typical moving window data is shown in Fig. 5.1

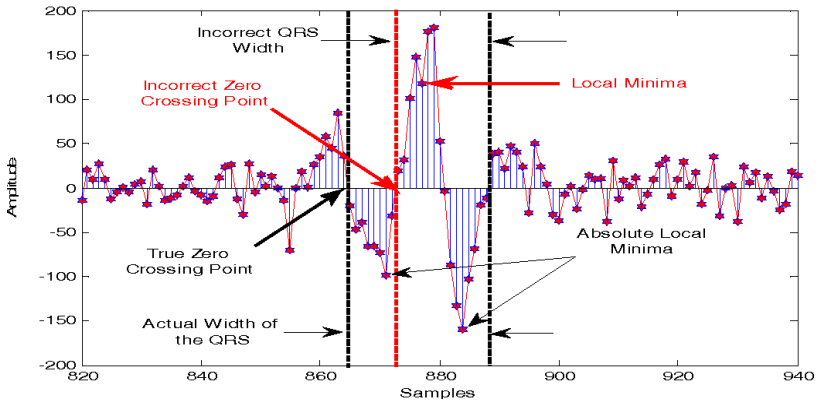


Figure 5.1: Moving Window

- Features are extracted from time windows and then represented as a vector
- Some features extracted are, mean, median, mode, standard deviation, minimum, maximum, Zero crossing -identify the number of times the time series data goes above the mean.
- The right featurization depends on the problem at hand.

5.4 Feature importance and Forward feature selection

Not all features of a model are equally important. Some faatures will be influencing the decision of a model more than other features. Filtering the important features will help us to reduce the dimension and speed-up the computation. Noise in the data may also get reduce as we are filtering out unwanted features.

- Feature importance helps to understand model interprettability.
- A classic KNN model doesnt give feature importance. There are some other models like Decision tree, Logistic regression which gives better feature importance.

5.4.1 Feature Selection

An efficient way to select important featueres of a model is with **backward feature elimination** method. Here we first train the model with all features and then eliminate features one by one in increasing order of importance of features until there is no significant drop in accuracy.

- This method helps to eliminate highly correlated features as the accuracy will not drop much if we try to remove a correlated feature.
- An alternative to this method is Forward feature selection which works in the opposite way. Its computationally more expensive.
- The time complexity of these methods is $O(d^2)$.
- This methods can be applied to any type of model.

5.5 Data preprocessing and column Scaling

A dataset is represented as follows, where columns represent the featues, each row represent one data point. Y can be represented as a column vector

$$D = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ N \end{array} \begin{array}{c} \begin{array}{ccccc} f_1 & f_2 & f_3 & \dots & f_d \end{array} \\ \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \dots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \dots & x_d^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_3^{(3)} & \dots & x_d^{(3)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ x_1^{(N)} & x_2^{(N)} & x_3^{(N)} & \dots & x_d^{(N)} \end{bmatrix} \end{array}$$

5.5.1 Min-Max Scaling on Columns

Preprocessing referes to the operations done on data matrix before feeding this into model. **Column scaling** is such a technique.

- Berfore dimensionality reduction we need to scale the data, because these techniques are *sensitive to scale of data.*
- By bringing all features into same scale, we can avoid the danger of having higher weightage to some features compared to other features.

Let $[x_1^i, x_2^i, x_3^i, \dots, x_N^i]$ be the i^{th} column of a dataset. The column is scaled using the following Eq. 5.6. After column normalization each value will lie in the interval $[0, 1]$.

$$x_j^{i\prime} = \frac{x_j^i - \min(x^i)}{\max(x^i) - \min((x^i)} \tag{5.6}$$

5.5.2 Column standardization

Column standardization is used more compared to column min-max scaling. It is used widely in gaussian distributed data.

- The formaula is similar to standrd univariate formula for gaussian distribution, but the difference is that the x_s need not be gaussian and it can come from any distribution.
- Geometrical intution of standardization: If data is of 2D, mean after standardization will be orign of XY plane. Standard deviation will be shifted to 1 on either side of mean.
- Column standardization is often called **mean centering and scaling** because it moves the mean to 0 and set the standard deviation to 1. The scaled value indicate how many standard deviation away is the value from mean in unscaled domain.

$$x_j^{i\prime} = \frac{x_j^i - \mu_j}{\sigma_j} \tag{5.7}$$

To do the column standardization for all features in one go, we need to compute **Mean vector of a dataset (mean vector)**. It is the vector which contains the mean of corresponding features.

5.6 Feature Projection Algorithms- PCA

An alternative approach to feature selection for dimensionality reduction is **feature projection**. In practice, feature projection is not only used to improve storage space or the computational efficiency of the learning algorithm, but can also improve the predictive performance by reducing the *curse of dimensionality* -especially if we are working with non-regularized models. Hughe's phenomenon says if higher the dimensions of a dataset higer the chances of overfitting.

Three major dimensionality reduction techniques by feature projection method are,

1. **Principal Component Analysis (PCA)** - unsupervised, linear
2. **Linear Discriminant Analysis (LDA)** - supervised, linear
3. **Kernel Principal Component Analysis (KPCA)** - unsupervised, nonlinear

In this section we discuss PCA method. PCA is fundamentally a **dimensionality reduction algorithm**, PCA can also be used for,

- Visualization of high dimensional data
- Noise filtering - Eigenfaces is an old technique for face recognition from images. Noise filtering is done by reducing the dimensions.

In a nutshell, PCA aims to find the directions of maximum variance in high-dimensional data and projects it onto a new axis (**principal components**). This concept is illustrated in Fig. 5.2, where x_1 and x_2 are the original feature axes, and PC1 and PC2 are the principal components. Note that, any two principal components are orthogonal.

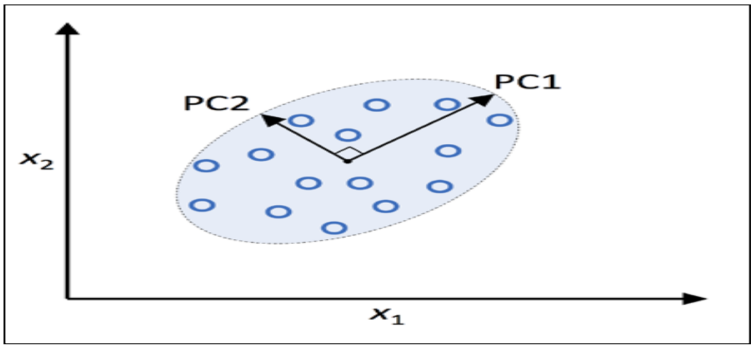


Figure 5.2: Principle Components

The two qunatities that the PCA fit on data and learns are:

1. **Principal Components :** Direction of the vector or priciple axes. Each dimension is associated with a principle component and each principle component has an explained variance associated with it. The first priciple component will have the higest variance followed by the second component and so on.
2. **Explained variance :** It is a measure of the variance of the data when projected onto that axis - indicates how "important" that axis is in describing the data.

The transformation from original data axes to principal axes is as an **affine transformation**, which is composed of:

1. Translation
2. Rotation
3. Uniform scaling

To reduce the dimensison of dataset D, we rank the principal components based on the explained variance. Higher the variance more important is that component in keeping the essence of orignal data. We zero out one or more principal components which has realatively lower explained variances.

- By looking at cummulative explained variance ratio plot, we can decide on the number of dimensions to keep.
- The variance ratio, helps us to understnad the level of redundancy present in the data set.
- While we eliminate the redundant principal components, we are keeping the signal and throwing out noise.
- Even if the Ds input features are correlated, the resulting principal components will be uncorrelated (mutually orthogonal).

Fig. 5.3 shows the idea of principle compoents, explained variance, cummulative expalined variance. We choose the new dimenion $k << d$, based on the plot.

- To find the principle components and corresponding explained variance, covariance matrix Σ of D is computed and decomposed to eign vectors u (principle components) & eigen values λ (explained variance).
- The eigenvalues define the magnitude of the eigenvectors, so we have to sort the eigenvalues by decreasing magnitude.

The **variance explained ratio** of an eigenvalue λ_j of principle component PC_j is given in Eq. 5.8.

$$\text{variance explained ratio} = \frac{\lambda_j}{\sum_{j=1}^{d} \lambda_j} \tag{5.8}$$

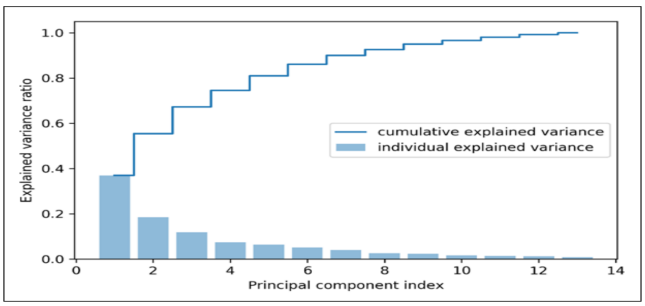


Figure 5.3: Choosing the number of components

The problem of PCA can be formulated in two different methods that are equilvallent. Both the methods involves the step of finding multiple axis that capture the original data in decreasing order of importance and then projecting the orignal data to those axes.

The first method tries to find the projection axis where the maximum variance for all data points can be found. The second method is based on minimizing distance between the data point and the projection axis. Both the methods are summarized in Fig. 5.4

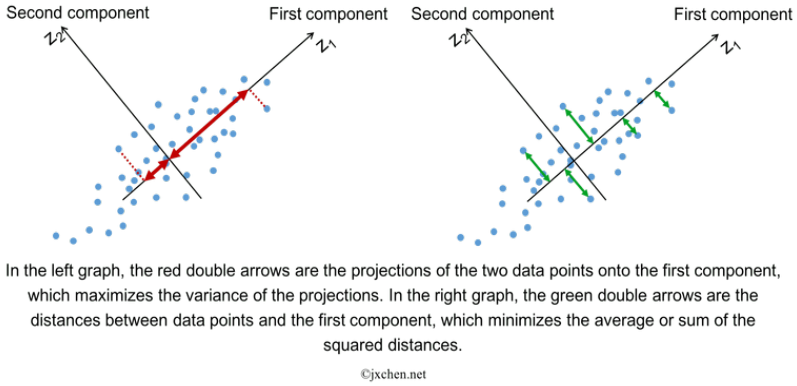


Figure 5.4: PCA Formulation

5.6.1 Method 1: Maximize projected variance

In this method, we find the **direction** (u) - a unit vector, on which maximum variance is present and then project the data onto that axes.

Let $x_i^{'}$ be the projection of x_i on u, i.e $x_i^{'} = proj_u(x_i) = \frac{u.x_i}{||u||}$, since $||u|| = 1$, we can write the equation as $proj_u(x_i) = u.x_i$. So the mathematical objective function becomes:

$$\text{Find } u \text{ such that} Var\Big\{proj_u(x_i)\Big\}_{i=1}^{N} \text{is maximal.}$$

$$Var\Big\{u^T x_i\Big\}_{i=1}^{N} = \frac{1}{N}\sum_{i=1}^{N}(u^T x_i - u^T \bar{x})^2 \tag{5.9}$$

PCA component calculations are highly sensitive to data scaling as we compute covariance of featuers, so its mandatory to scale the columns. If we do standard scaling on the features, the mean vector $\bar{x} = 0$.

$$Var\Big\{u^T x_i\Big\}_{i=1}^{N} = \frac{1}{N}\sum_{i=1}^{N}(u^T x_i)^2 \tag{5.10}$$

So the final objective function becomes a **constraint optimization** problem

$$\max_u \Big\{\frac{1}{N}\sum_{i=1}^{N}(u^T x_i)^2\Big\} \text{ subject to } ||u|| = 1 \tag{5.11}$$

Eq. 5.11 can be rewritten as an easy to implement Eq. 5.12, where Σ is the covariance matrix of dataset D with features X.

$$\begin{aligned} max_u u^T \Sigma u \qquad\qquad & \text{s.t } u^T u = 1 \\ \Sigma = \frac{1}{N} X^T X & \end{aligned} \tag{5.12}$$

For solving this optimization problem the Lagrange multiplier equations can be written as

$$L(u, \lambda) = u^T \Sigma u - \lambda\{u^T u - 1\} \tag{5.13}$$

$$\frac{\partial L}{\partial u} = 2\Sigma u - 2\lambda u \tag{5.14}$$

$$\frac{\partial L}{\partial u} = 0 \Rightarrow \Sigma u = \lambda u \tag{5.15}$$

- The solution to PCA optimization problem is nothing but the **eigen vectors** u & **eigen values** - λ, as given in Eq. 5.16

$$\Sigma u = \lambda u \tag{5.16}$$

Also λ is scalar, $u_i \perp u_j, \forall i \neq j$.

5.6.2 Method 2: Minimize projected distance

For each data point x_i we can get the distance d_i from projection axis u_i. The objective function here is to minimize the squared distance of d_is of all data points.

$$\min_u (\sum_{i=1}^{N} d_i^2) \tag{5.17}$$

Lets try to find a formula for distances d_i. From the Fig. 5.4 , we can see that the distance d_i can be computed using pythogorus theorem as

$$\begin{aligned} d_i^2 &= ||x_i||^2 - (u^T x_i)^2 \\ &= (x_i^T x_i) - (u_1^T x_i)^2 \end{aligned} \tag{5.18}$$

The final formula becomes

$$\min_u \Big(\sum_{i=1}^{N} (x_i^T x_i) - (u^T x_i)^2 \Big) \text{ with a constraint } u^T u = 1.$$

5.6.3 PCA summary

It can be proved that the u which maximizes the variance in Method 1 is same as the u which minimizes the distance in Method 2 described above. PCA steps can be summarized as follows,

1. Column standardize the d-dimensional dataset D. $\mu = 0, \sigma = 1$
2. Construct the covariance matrix - Σ of D
3. Decompose Σ into its eigen vectors u and eigen values λ.
4. Rank the u_is by decreasing order of corresponding λ_is.
5. Select top k eigen vectors & eigen values, where k is the dimensionality of the new feature subspace ($k \leq d$). k has to be choosen based on the explained cummulative variance ratio.
6. Construct a projection matrix W of dimenison $d \times k$ from the "top k" eigen vectors.
7. Project or transform D using the projection matrix W to obtain the new k-dimensional transformed dataset D' by matrix multiplication. $D' = D.W$. D' is a $N \times k$ data set.

5.6.4 Limitations of PCA

PCA basically tries to preserve global structure of data. PCA doesnt consider distance between data points rather nor it tries to retain the local structures like clusters in the data.

- PCA does not perform well when there are nonlinear relationships within the data. It will not be able to interpret complex polynomial relationship between features.

- PCA tends to be highly affected by outliers in the data.
- If data is spread like circle ($2D$), sphere ($3D$) or hypersphere (nD), if we use PCA, we are going to loss lot of information.
- If data is forming groups or clusters in the original spcace projecting them will cause loosing of information about clusters in the dataset.
- If data has a distribution like sinusoidal, after projection we loose the shape information.

5.7 Curse of Dimensionality

High dimensionality causes problems in ML model training. Three areas where high dimensionality causes the problems listed below,

1. **Hughes phenomenon** which states,
   - With fixed number of data points the performance of ML model decreases as we increase the number of dimensions.
   - So it is important to reduce the dimension or increase the training set size before we build an ML model.
2. **Euclidean distance**
   - If we draw some bunch of points randomly from very high dimensional space, the Eulcidean distance between any two points is roughly the same.

$$\lim_{d\to\infty, i\neq j} \frac{max\_dist(x_i, x_j) - min\_dist(x_i, x_j)}{min\_dist(x_i, x_j)} = 0 \qquad (5.19)$$

   - This problem can influence the models which uses Euclidean distance such as KNN, K-means etc.
   - We prefer to use cosine similarity in high dimension. Typically sparse data cosine similarity works well in high dimension for NLP related tasks.
3. **Overfitting**
   - High dimension can easily result the model to overfit.
   - We can use PCA or other dimensionality reduction methods to reduce the impact of high dimensions.

6

Visualize Your Data

In this section, we study graphic displays that are helpful for the visual inspection of data that helps in data preprocessing. These displays include

- Histograms (Univariate Plot)
- Quantile plots (Univariate Plot) & Quantile-Quantile plots (Bivariate Plot)
- Scatter plots (Bivariate Plot)
- Box plots (Univariate Plot)
- Violin plots (Univariate Plot)
- Contour Plots (Bivariate Plot)

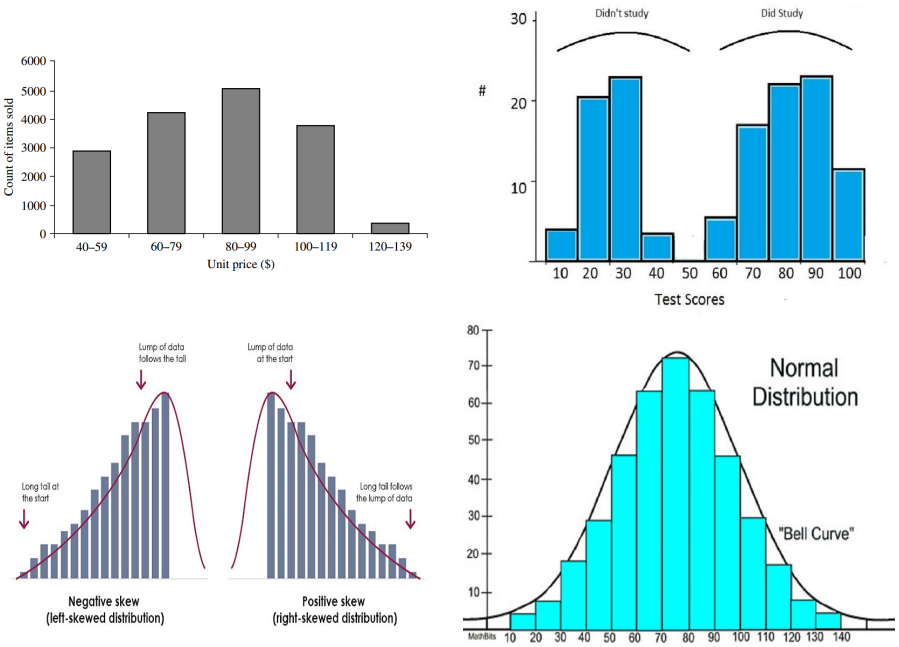
6.1 Histograms

"Histos" means pole, and "gram" means chart, so a histogram is a chart of poles. Plotting histograms is a graphical method for summarizing the distribution of a given attribute, X.

- If X is nominal, then a pole or vertical bar is drawn for each known value of X.
  - The height of the bar indicates the frequency (i.e., count) of that X value.
  - The resulting graph is more commonly known as a **bar chart**. eg: *model_type* or *item_type*
- If X is numeric, the term **histogram** is preferred.

– The range of values for X is partitioned into disjoint consecutive subranges called textbfbuckets or **bins**. Typically, the buckets are of equal width. eg: *Price data* $10 - 20$.

– Histograms can be converted into probability function by smoothing method done using Kernel density estimation.



PDF is smoothed version of histogram.

- It is called density function because each region shows how many points are there in the region (density).
- We can use PDF plot to check the separability of clasess for each of the feature. If density function is well separated along X-axis, then that feature is more important in classifying data points.

6.2 Quantile Plot & Quantile–Quantile Plot

Quantiles are values that split sorted data or a probability distribution into equal parts. In general terms, a k-quantile divides sorted data into k parts. On a quantile plot, a column value is graphed against its percentile. Fig. 6.1 shows a quantile plot for sales data.

In statistics, (quantile-quantile) Q-Q plots play a very vital role to graphically analyze and compare two probability distributions by plotting their quantiles against each other. Fig. 6.2 shows a QQ-plot.

Q-Q plot can be used to check whether two random variables X, Y follows the same distribution. There are three major steps in $Q - Q$ plot.

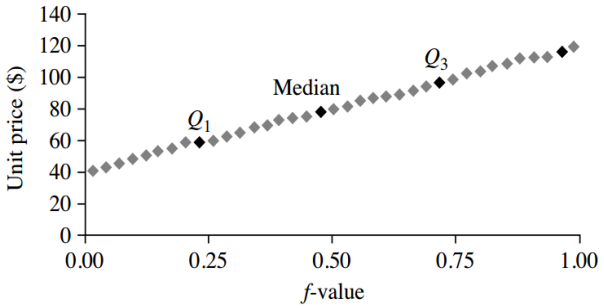


Figure 6.1: Quantile plot

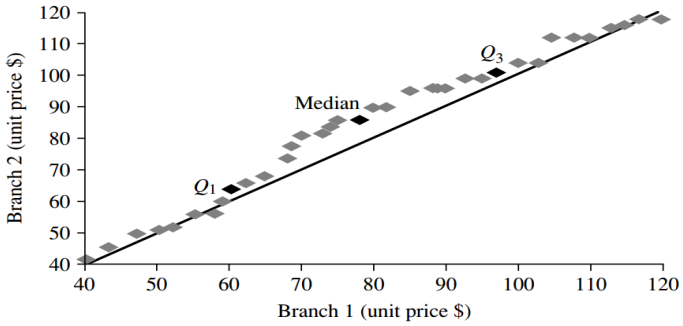


Figure 6.2: quantile-quantile plot

1. Get percentile values of X, by sorting and picking $1^{st}, 2^{nd}, \ldots, 100^{th}$ percentile values.

2. Generate percentile values of random variable Y that follows some distribution (Gaussian, uniform, poisson etc.).

3. Plot Y v/s X

   - If the plot looks similar to line $(Y = X)$ then we can conclude that X, Y follows the same distribution.
   - If the points at the ends of the curve formed from the points are not falling on a straight line but indeed are scattered significantly from the positions then the two distributions are different.
   - We can check the given distribution X follows Gaussian distribution, by setting $N(0, 1)$ as Y and QQ-plotting.

4. The number of points in X, Y distribution should be sufficently large, otherwise its difficult to inteprett the QQ-plot.

5. Three applications of QQ-plot are

- Compare two distributions are same or not
- Check Skewness in distribution
- Check for peakedness or tailedness of a distribution

Fig. 6.3 shows QQ-plot for checking a sample distribution follows normal distribution or not.

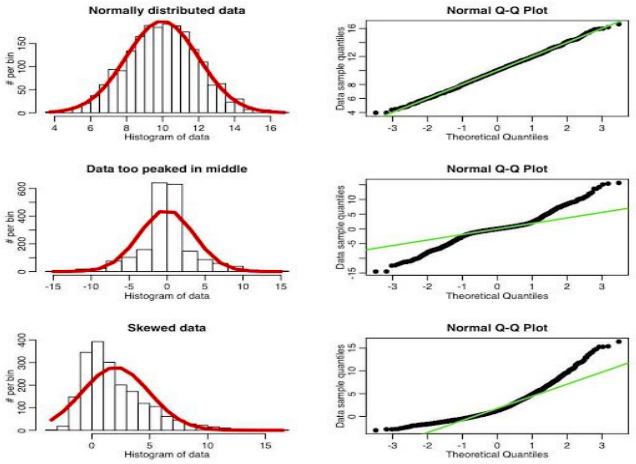


Figure 6.3: QQ plot for Normality check

Fig. 6.4 shows QQ plot for checking a sample distribution is positively skewed or neagtively skewed.

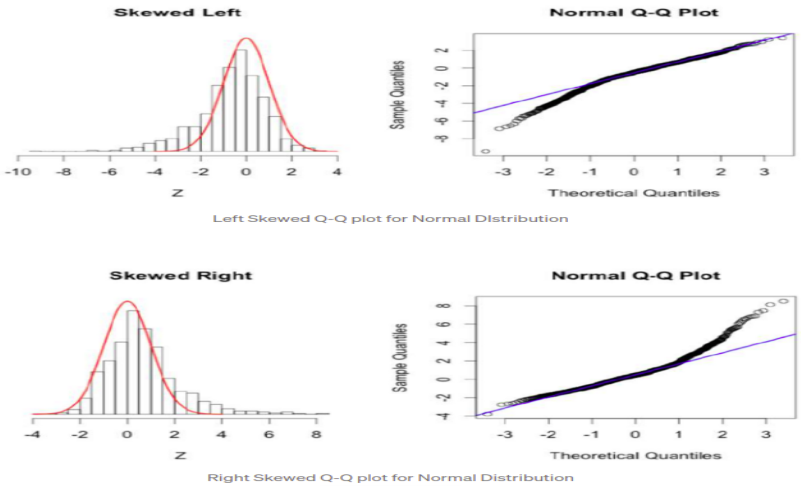


Figure 6.4: QQ plot for skewness check

- If the bottom end of the QQ-plot deviates from the straight line but the upper end follows straight line, then we can clearly say that the distribution has a longer tail to its left or simply it is left-skewed (or negatively skewed).
- When we see the upper end of the QQ-plot to deviate from the straight line and the lower and follows a straight line then the curve has a longer tail to its right and it is right-skewed (or positively skewed).

Fig. 6.5 shows QQ-plot for checking a sample distribution has fat tail or thin tail.

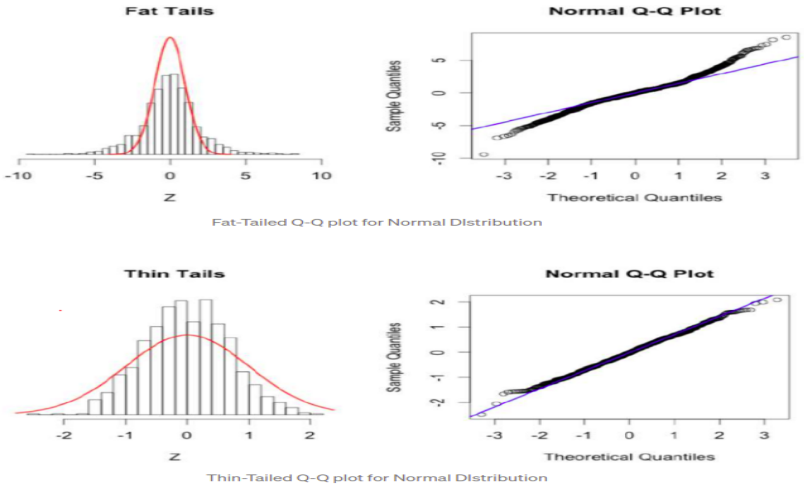


Figure 6.5: QQ plot for tailedness check

- The distribution with a fat tail will have both the ends of the QQ-plot to deviate from the straight line and its center follows a straight line.
- A thin-tailed distribution will form a QQ-plot with a very less or negligible deviation at the ends.

6.3 Scatter Plots and Data Correlation

A **scatter plot** is a graphical method for determining if there appears to be a relationship, trend between two numeric attributes. Fig. 6.6 shows a sample scatter plot.

- To construct a scatter plot, each pair of values is treated as a pair of coordinates in an algebraic sense and plotted as points in the plane.
- The scatter plot is a useful method for providing a first look at bivariate data to see:

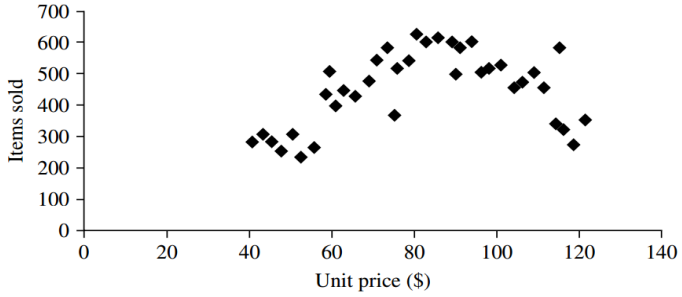


Figure 6.6: A scatter plot for data set.

  - Clusters of points
  - Outliers
  - Explore the possibility of correlation

- A line of best fit can be drawn to study the correlation between the variables. Two attributes, X, and Y, are correlated if one attribute implies the other. Correlations can be positive (+ slope), negative (− slope), or null (uncorrelated).
- Scatter plots can be extended to n attributes, resulting in a **paired scatter-plot**.

Fig. 6.7 shows three cases for which there is no correlation between the two attributes.

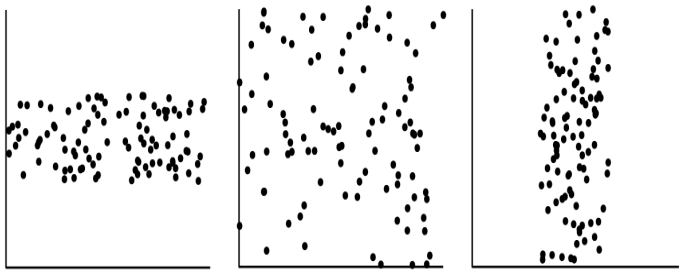


Figure 6.7: Three cases where there is no observed correlation

6.4 Boxplots

A box plot shows data from a five-number summary with quartile information ((**Minimum,** Q_1**, Median-**Q_2**,** Q_3**, Maximum**)). It does not show

the distribution in particular as much as a histogram or PDF plot does. Fig. 6.8 shows a boxplot and it's terms.

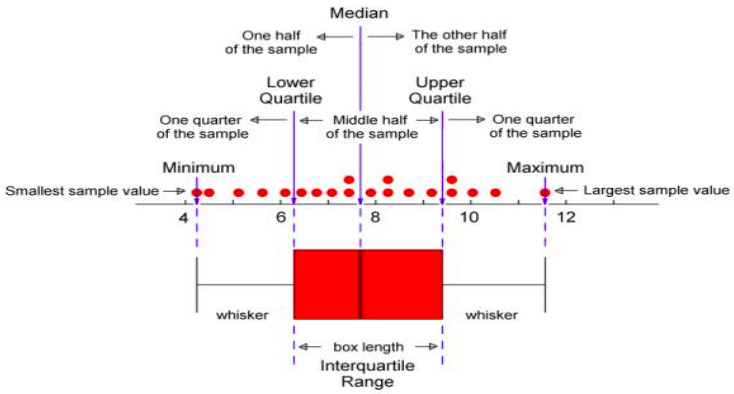


Figure 6.8: Boxplot

A box plot is primarily used for

- Check symmetry of data
- Spot potential outliers
- Check the skewness and the direction of skewness

We can check the box plots tail length, position of median, presence of dots to get an idea of symmetric distrbution, skewness & its direction and the outliers.

- **Symmetric**: The box plot is said to be symmetric if the median is equidistant from the maximum and minimum values.
- **Outliers**: Potential outliers will be present as dots in the plot
- **Skewness**:
  - **Negatively Skewed**: If the distance from the median to minimum is greater than the distance from the median to the maximum, then the box plot is negatively skewed.
  - **Positively Skewed**: If the distance from the median to the maximum is greater than the distance from the median to the minimum, then the box plot is positively skewed.
- More than one variables can be compared side by side using parallel box plots - one box plot for each variable

- Sometimes box plots can be misleading. They are not affected by data's distribution. When the data "morph" but manage to maintain their stat summaries (medians and ranges), their box plots stay the same.

6.5 Violin plots

Violin plots are used when you want to observe the distribution of numeric data, and are especially useful when you want to make a comparison of distributions between multiple groups. A violin plot is shown in Fig. 6.9

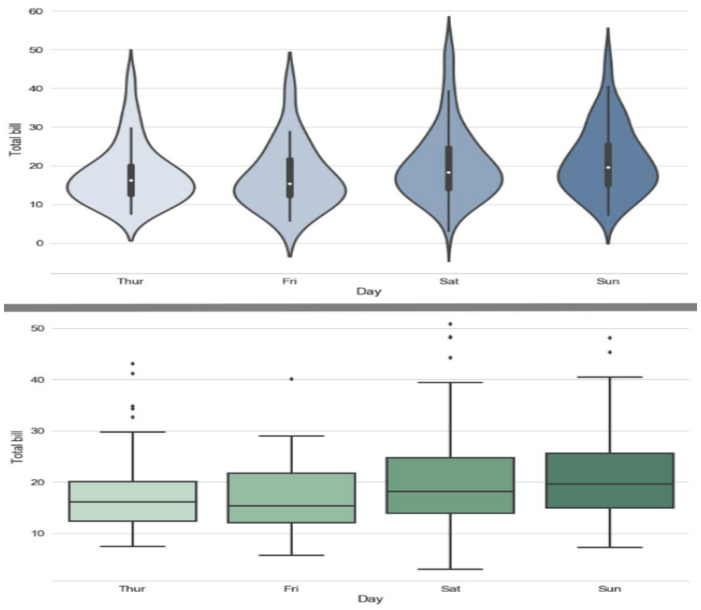


Figure 6.9: Violin Plot

- As we combine two plots in violine plot, we get probability distribution (PDF/Histogram) as well as the quartile information (box plot) in one plot.

- The "violin" shape of a violin plot comes from the data's density plot. The distribution on either side of the boxplot information in violine plot are mirror images of each other.

- Thicker part of violin means the values in that section has higher frequency, and the thinner part implies lower frequency.

- Violin part of plot helps to check the data is unimoal or multimodal.

- Why don't we just use a density plot instead of violin plot? - When there are too many groups (more than 3), their overlapping density plots become difficult to read.
- Unlike box plot, we will be able to see the changes in data even if the quartile values remains same. PDF part in violin plot helps to detect such changes.

6.6 Contour plots

Contour plots are a way to show a $3D$ surface on a $2D$ plane - it is an alternative to a $3D$ surface plot. A contour plot is appropriate if you want to see how some value Z changes as a function of two inputs, X and Y: $z = f(x, y)$.

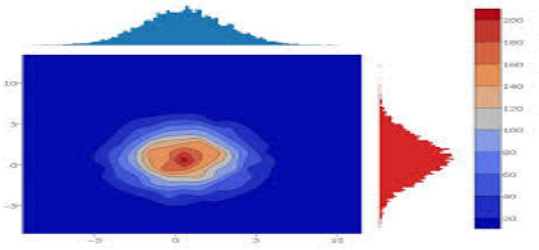


Figure 6.10: Contour plot

- PDF shows 1D density. Contour plots shows a 2D density plot of two continuous random variables.
- The two axes reprresent the density of two random variables and the points in the plane represents the aggregated density of both variables.
- In machine learning, contour plots are used to analyze the global minimum points in backpropagation training.

6.7 t-SNE (t-distributed stochastic neighbourhood embedding)

t-SNE helps to visualize the high dimensinal data in lower dimensions like $2D$ & $3D$. It is a unsupervised randomized method that reduces the high dimension to lower dimension. In contrast to PCA, t-SNE reduces the high dimension of data in a non-linear fashion and also preserves the local structure in the high dimensional data using student t-distribution.

It tries to keep similar data points closer in lower dimension and dissimilar ones as apart as possible to retain local and global structures in the data. It can be used only for data visualization to uncover hidden complex relationship between data points.

- In higher dimension : t-SNE computes pairwise similarity of points and map those distances to conditional probabilities using normal distribution. It computes a similarity matrix S_h for high dimension.
- t-SNE then arranges these high dimensional points to lower dimension.
- In lower dimension : t-SNE computes pairwise similarity of points and map those distances to conditional probabilities using t-distribution. It computes a similarity matrix in lower dimension S_l. Without t-distribution, the clusters would all clump up in the middle and will be harder to visualize.
- t-SNE compares the similarity matrices S_h & S_l and tries to minimize the difference such that all the pairs have a similar probability distribution (both in higher and lower dimension). Gradient Descent is used with Kullback Leibler Divergence between the two distributions as a cost function

t-dsitribution helps to minimize the **crowding problem** - i.e. in some cases it is impossible to maintain the neighbourhood distance in lower dimension. eg. if our orignal data is $2D$ and all datapoints belong to four corners of a square, we cannot find the embedding in 1D keeping the distance same. t-SNE has a computational complexity of $O(N^2)$ due to pairwise distance computatiion. So this doesn't scale well for large data sets. Fig. 6.11 shows t-SNE applied on MNIST dataset.

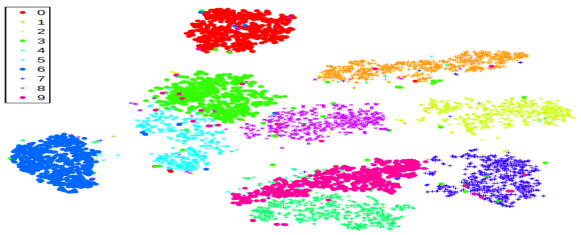


Figure 6.11: t-SNE visualization

6.8 Measures of Shape of Distributions

There are many data analysis tasks that assume the data is normally distributed. By looking into the histogram/distribution curve of data, we can

get basic idea about the shape of the distribtion. There are three things we notice when we look into a distribution plot,

1. Distribution is unimodal or multimodal
2. Distribution is skewed or not
3. How tall/sharp is the peak of the distribution

Two heuristic methods that helps to quantify shape of distributions and test for normality condition are,

1. **Skewness coefficient:** Tells us the amount and direction of skew (deviation from horizontal symmetry).
2. **Kurtosis :** Explains how often observations in some dataset fall in the tails v/s the center of a probability distribution.

6.8.1 Measuring Skewness

A univariate distribution can be symmetric, positively skewed or negatively skewd as shown in Fig. 6.12.

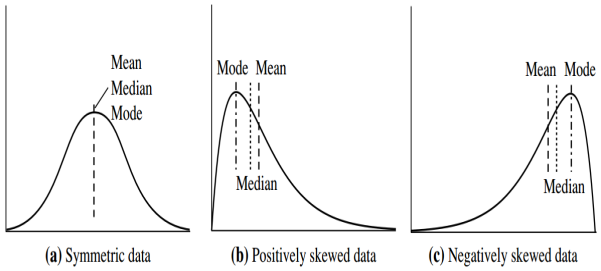


Figure 6.12: Symmetry and Skewness

- If the tail of distribution is longer in right side, we say that the distribution is **skewed right** or **positively skewed**.
  - The variance is more on right side of the mean. It is likely to find some extremely large values in my data which would be much larger than the mean and median.
- If the tail of distribution is longer in left side, we say that the distribution is **skewed left** or **negatively skewed**.
  - The variance is more on left side of the mean. It is likely to find some extremely small values in my data which would be much smaller than the mean and median.

The skewness of a data set is measured using **moment coefficient of skewness** using Eq. 6.1. It has no units: it's a pure number, like a z-score.

$$G_1 = \frac{m_3}{m_2^{\frac{3}{2}}}, \qquad \text{where} \tag{6.1}$$

$$m_3 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^3 \qquad \text{The third moment}$$

$$m_2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 \qquad \text{The second moment - Variance}$$

Skewness coefficient interpretation	
$G_1 = 0$	perfectly symmetric
$G_1 > 0$	Positively skewd - higher the value more is the skewness
$G_1 < 0$	Negatively skewd - higher the value more is the skewness

6.8.2 Kurtosis

Kurtosis is a measure of **tailedness** of the distribution. It is often measured in comparing with the normal distributions tailedness. Normal distribtion has a kurtosis of 3 - as 99.7% values lies witin 3 standard deviations from mean. The kurtosis, like skewness, has no units: Kurtosis lies in the interval $[1, \infty]$.

- Kurtosis measures the extream values in either tail, where as the skewness coefficient differentiate the extream values in left vs right tails. Thus the outliers in a sample, have more effect on the kurtosis than they do on the skewness.
- An increase in kurtosis is associated with the movement of *probability mass* from the shoulders of a distribution into its center and tails.

The moment coefficient of kurtosis (A_4) of a data set is computed using the Eq. 6.2, where m_4 is the fourth moment & m_2 is the second moment

$$A_4 = \frac{m_4}{m_2^2} \tag{6.2}$$

$$\text{excess-kurtosis} = A_4 - 3 \tag{6.3}$$

- An **excess kurtosis** is simply kurtosis-3. A normal distribution has an excess kurtosis of 0. In finance and investing, excess kurtosis is interpreted as a type of risk, known as "tail risk" or the chance of a loss occurring due to extream outliers.

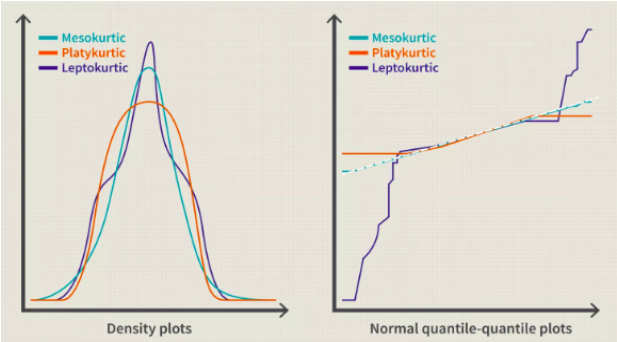


Figure 6.13: Types of Kurtosis

Kurtosis coefficient interpretation	
$A_4 = 3$ (Meoskurtic)	tails are similar to normal distribution
$A_4 > 3$ (Leptokurtic)	tails are longer and fatter compared with normal distribution (exceeds noraml distribution tails)-implies, most often we see extream outliers
$A_4 < 3$ Platykurtic)	tails are shorter and thinner compared with normal distribution (behind the tails of normal distributions)-implies, less often we see extream outliers

- Kurtosis is sometimes confused with a measure of the peakedness. A distribution can be infinitely peaked with low kurtosis, and a distribution can be perfectly flat-topped with infinite kurtosis.

7

Model Evaluation, Selection & Deployement

Model evaluation aims at estimating the performance of a model using a metric. Selecting the final model is done based on an evaluation metric. Models can also be compared with respect to the following additional aspects

- **Speed**
- **Robustness**
- **Scalability**
- **Interpretability** This refers to the level of understanding of the reason for prediction. In most of the cases we would like to know why the model has predicted so eg: why the model predicted the patient has this particular disease.
  - Models based on decision tree are easier to interpret, where as neural network based models are typically complex and difficult to intepret.
  - If we are able to get some reasoning from a model, such models are called the **interpretable models** otherwise its called **black box model**.

In this chapeter, we discuss various metrics used for evaluating models.

7.1 Evaluating Classifier Models

In this section we discuss about various metrics used to assess the performance of classification models.

7.1.1 Accuracy, Precision, Recall

We can categorize the samples in terms of **positive tuples** (tuples of the class of interest) and **negative tuples** (all other tuples - considering all other clasess put together).

There are four "building blocks" used in computing many evaluation measures for classification problems.

- **True positives (TP) :** These refer to the positive tuples that were correctly labeled by model
- **True negatives (TN) :** These are the negative tuples that were correctly labeled by model
- **False positives (FP) :** These are the negative tuples that were mislabeled as positive by model
- **False negatives (FN) :** These are the positive tuples that were mislabeled as negative by model

The total number of tuples in test set $= TP + FP + TN + FN$. These terms can be summarized into a confusion matrix as shown in Fig. 7.1.

Predicted class

Actual class	Sit	Stand	Walk Jog	Ascend	Descend	Cycle	**Class-specific recall**
Sit	*3202*	*2*	*0*	*0*	*0*	*14*	**0.99**
Stand	*7*	*3191*	*2*	*7*	*0*	*0*	**0.99**
Walk	*0*	*0*	*10647*	*74*	*0*	*0*	**0.99**
Ascend	*0*	*0*	*34*	*500*	*15*	*1*	**0.90**
Descend	*0*	*0*	*41*	*60*	*405*	*0*	**0.80**
Cycle	*146*	*3*	*0*	*0*	*0*	*2539*	**0.94**
Class-specific precision	**0.95**	**1.00**	**0.99**	**0.78**	**0.96**	**0.99**	**0.98**

Figure 7.1: Confusion Matrix

- The confusion matrix shows the distribution of predictions made by model. - basically it tells how well the classifier can recognize instances of each class.

The **accuracy** of a classifier on a given test set is the percentage of tuples that are correctly classified by the classifier.

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.1}$$

We can also speak of the **misclassification rate**

$$\text{misclassification rate} = \frac{FP + FN}{TP + TN + FP + FN} \tag{7.2}$$

We now consider the **class imbalance problem**, where the main class of interest is rare. For eg. If we consider the dataset with uses who have clicked an ad versus users who have not clicked an ad. Here the class of interset is the clicked users. Typically the number of users who click an ad is very less when compared with number of users who doesn't click an ad.

- Accuracy doesn't work when there is class imbalance. Model can easily get high accuracy by predicting all instances as negative class.
- As the negtive class is very common, if we measure accuracy it shows up high value and leads to wrong conclusion that model is good
- We need a metric that can very well predict the positive class

The **sensitivity**, **specificity**, **precision** measures can be used to assess model performance on postive class prediction.

- **Sensitivity** or **true positive rate** or **recall** : The proportion of positive tuples that are correctly identified. It is a measure of *completeness*
  - Recall is important when we need to improve the website conversion rate of users in ad-campaign. Loosing any user who is going to convert will lead to reduced revenue.
- **Precision :** The proportion of the predicted positive tuples that are correctly identified. It can be thought of as a measure of *exactness*
  - Precison is very important in case of information retrieval. eg: Based on a search query, we retrieve documents and the retrieved documents should be relevant to the query.
- **Specificity** or **true negative rate** : The proportion of negative tuples that are correctly identified.

The sensitivity and specificity can be measured as follows:

$$\text{sensitivity} = \frac{TP}{TP + FN} \tag{7.3}$$

$$\text{precision} = \frac{TP}{TP + FP} \tag{7.4}$$

$$\text{specificity} = \frac{TN}{TN + FP} \tag{7.5}$$

- Model can get 100% recall by predicting everything as postive class instances
- Model can get 100% precision by predicting only one postive instance correct
- So, Neither recall nor precision alone can judge the performace of model

For solving the above pitfalls with single metric like precision and recall, we can think about combining these two metrics - this is the approach of the **F measure** (also known as the **F1 score** and F_β measure, where β is a non-negative real number.

- The f1-score is the **harmonic mean** of precision & recall. As harmonic mean of two numbers will be always closer to the minimum, f1-score penalize more if any of the precesion or recall is low.
- The F_β-score is a weighted harmonic mean of precision and recall. It assigns β times as much weight to recall as to precision.

$$F1\text{-score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{7.6}$$

$$F_\beta\text{-score} = \frac{(1 + \beta^2) * \text{Precision} * \text{Recall}}{\beta^2 \text{ *Precision} + \text{Recall}} \tag{7.7}$$

7.1.2 Micro avergae F1-score and Macro Average F1-score

How do we evaluate the performance of **multi-labelled classifirer**. multi-labelled classification referes to the case where an instance can be assigned to more than one class and there are many clasess available. for eg: stack-overflow query posted by a user has 3 tags C, C++ & Python.

Lets take an example that has maximum 3 labels. Table 7.1 shows this and Table 7.2 shows the encoding for clasess in the order A, B, C.

Table 7.1: Multi-labelled classification

Instance #	Target	Predicted
1	A, C	A, B
2	C	C
3	A, B, C	B, C

Let's find the confusion matrix for class A based on the above prediction results and encoding table. The prediction numbers for class A are defined below.

Table 7.2: Multi-labelled encoding

Instance #	Target	Predicted
1	1 0 1	1 1 0
2	0 0 1	0 0 1
3	1 1 1	0 1 1

- TP : Predicted class labels has A in it & the actual label set has A
- TN : Predicted class labels has no A in it & the actual label set has no A
- FP : Predicted class labels has A in it but the actual label set
- FN : Predicted class labels has no A in it but actual label set

Based on this idea, we can calculate the confusion matrix for each clasess and compute metrics like precesion, recall, f-score etc like the way do for a binary classification problem. Confusion matrix for class A is given below.

		Predicted		
		Negative	Positive	Total
Actual	Negative	1	0	1
	Positive	1	1	2
	Total	2	1	3

We need to combine the metrics from each confusion matrix and finally come up with a single metric. So the aggregation of results can be done based on two methods

- **macro averaging** : We compute the precision, recall, f-score for each clasess and average them
- **micro averaging** : We sum up the confusion matrices for all clasess and form the final confusion matrix, then the precision, recall, f-score are computed from the final confusion matrix.

7.1.3 Hamming Loss for Multilabelled Classification

Another method to evaluate the multi-lablelled setting is using the **Hamming loss**. Here we represent the predicted class label for an instance y_i as a C dimensional binary vector, where C is the number of clasess. Hamming loss is computed as follows,

$$H = \frac{1}{N} \frac{\sum_{i=1}^{N} \text{XOR}(y_i, \hat{y}_i)}{|C|} \tag{7.8}$$

7.1.4 Cost–Benefit and ROC Curves

In some cases the cost associated with a FN (such as incorrectly predicting that an actual patient as not patient) is far greater than those of a FP. These costs may consider the danger to the patient, hospital reputation etc. In such cases, we can outweigh one type of misclassification over another by assigning a different cost to each.

An **Receiver operating characteristic** curve (ROC curve) is a graph showing the performance of a classification model at different classification thresholds. This curve plots two parameters:

- **True Positve Rate** on Y-axis: Rate at which a positive sample is correctly classified
- **False Positive Rate** on X-axis: The rate at which a negative sample is missclassifed

Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. Fig. 7.2 shows a typical ROC curve.

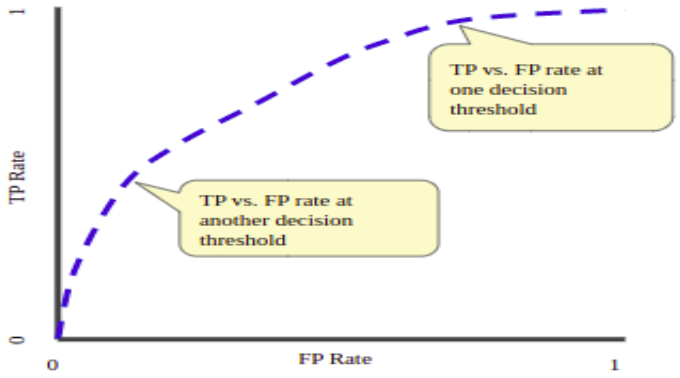


Figure 7.2: ROC Curve

To compute the points in an ROC curve, we could evaluate a logistic regression model many times with different classification thresholds, but this would be inefficient. Fortunately, there's an efficient, sorting-based algorithm that can provide this information for us, called **Area Under the Curve** - AUC.

- AUC provides an aggregate measure of performance across all possible classification thresholds.
- One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example.

- AUC value can be read as follows:
  - AUC = 1.0 => ideal classifier
  - AUC = 0.0 => worst classifer
  - AUC = 0.5 => random classifer
  - AUC > 0.5 => a typical classifier
  - AUC < 0.5 => a poor classifier

Fig. 7.3 shows the probabilistic interpretation of AUC and Fig. 7.4 shows an AUC curve.

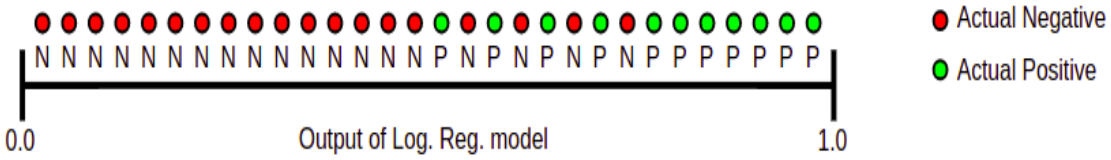


Figure 7.3: ROC Curve

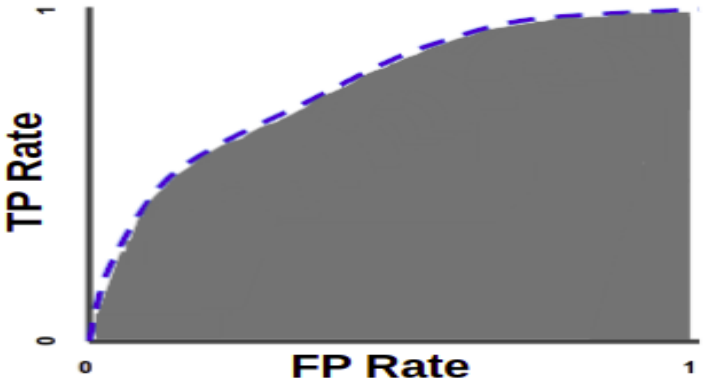


Figure 7.4: ROC Curve

The benefit and pitfalls of AUC can be summarized as follows

- **scale invariant** : It measures how well predictions are ranked, rather than their absolute values
  - sometimes we really do need well calibrated probability outputs, and AUC won't tell us about that. So in that case AUC is not appropriate.
- **classification-threshold-invariant** : It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

  - For, email spam detection, you likely want to prioritize minimizing false positives (even if that results in a significant increase of false negatives). AUC isn't a useful metric for this type of optimization

7.1.5 Entropy

Entropy in physics/chemistry referes to the degree of randomness. Entropy in machine learning refers to the randomness or impurity in features- it characterize the "unpredictability" of a random variable. Entropy sets the foundation for:

- Decsion tree based algorithms
- KL-diverence idea used in t-SNE
- Quantify similarity & differences between two things

Before we dive into entropy, we should first understand **surprise**. Lets take two types of coin tossing cases.

1. Fair coin : Head & Tail are equally likely with probability 0.50 for each
2. Biased coin : Head is more likely with probability 0.95 and tail very unlikely with probability 0.05

In the coin toss experiment, if we toss an fair coin and if we get head or tail we won't be *surprised*, but when we toss biased coin and get a tail. So we define the surprise as a measure to quantify seeing the outcome. While quantifying a surprise, we should consider the following

- Surprise of an event is inversly proportional to probability of that event
- Surprise is 0 when the probability for the event is 1
- Surprise is ∞ when probability for the event is 0

Now lets look into a method for quantifying surprise (S_i) of an event i, given probability of that event is p_i. Lets take an edge case where probability is 1.

- We want the surprise to be 0 when we have probability = 1.0.
- We can not directly take $\frac{1}{p_i}$ as this gives 1, rather we take $\log \frac{1}{p_i}$ which is equal to 0 when $p_i = 1$

- Surprise for any event like flipping coin 3 times and getting HHT can be computed by first computing the probability of that event and substituting in the equation.

What is the average surprise we get if we toss the coing 100 times. This average surprise of an event is nothing but the **entropy**

- Entropy of an event is the expected value of surprise

Entropy of many events is estimated by takig the expected value of each event's surprises. Fig. 7.5 shows the entropy concept.

$$H = \sum_i p_i * S_i \tag{7.9}$$

$$H = \sum_i p_i \log\left(\frac{1}{p_i}\right) \tag{7.10}$$

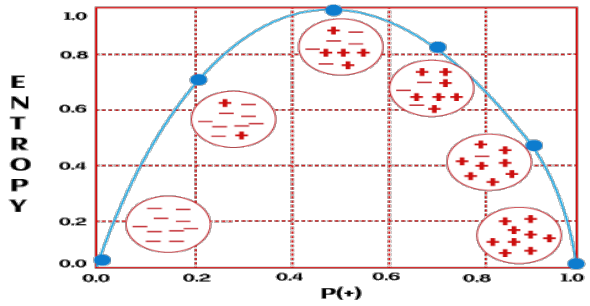


Figure 7.5: Entropy Concept

- Decision tree based algorithm uses entropy to split a feature into branches. If higher the entropy more useful is that feature in learning.

- Other way of interpretting the entropy is to identify the minimum number of bits required to encode an information. Suppose p_i is the probability of the i^{th} symbol in our information,

  - smallest number of bits required to encode the information would be: $\log\left(\frac{1}{p_i}\right) = -\log p_i$
  - Entropy is the expectation of number of bits required for each event

7.1.6 Cross Entropy

If we think of a distribution as the tool we use to encode symbols, then entropy measures the minimum number of bits we'll need if we use the correct tool y (ground truth).

- In contrast, **cross entropy** is the number of bits we will need if we encode symbols from y using the wrong tool $\hat{y}$.
- This consists of encoding the i^{th} symbol using $\log \frac{1}{\hat{y}_i}$ or $\log \frac{1}{p_i}$ bits instead of $\log \frac{1}{y_i}$ bits.

We of course still take the expected value to the true distribution y , since it's the distribution that truly generates the symbols:

$$H(y, \hat{y}) = \sum_i y_i \log\left(\frac{1}{p_i}\right) \tag{7.11}$$

- Cross entropy is always greater than entropy. We use cross entropy as a loss function to minimize for classification problems.
- In cross entropy loss, y represents the ground truth label (0 or 1) and $\hat{y}$ represents the probability of an instance belonging to a class as predicted by model (predicted probability).
- Cross entropy for an instance would be 0 if model predicted it correctly with probability 1, non-zero otherwise.

7.1.7 KL Divergence

The KL divergence from $\hat{y}$ to y is simply the difference between cross entropy and entropy,

$$H(y||\hat{y}) = \sum_i y_i \log\left(\frac{1}{\hat{y}_i}\right) - \sum_i y_i \log\left(\frac{1}{y_i}\right) = \sum_i y_i \log\left(\frac{y_i}{\hat{y}_i}\right) \tag{7.12}$$

- It measures the number of extra bits we'll need on average if we encode symbols from y according to $\hat{y}$.
- It's never negative, and it's 0 only when y and $\hat{y}$ are the same.
- Minimizing cross entropy is the same as minimizing the KL divergence from $\hat{y}$ to y, since entropy is a constant here.

7.1.8 Log Loss Function

Log loss uses the actual probability scores. Its a cost function for logistic regression model. The Log loss is computed using the Eq. 7.13.

$$log - loss = -\frac{1}{n}\sum_{i=1}^{N} y_i \log p_i + (1 - y_i) \log (1 - p_i) \tag{7.13}$$

- Log loss can be defined as the average logarithm of probability of being the correct class label.
- Log loss will penalize a lot for small loss in probability. We want log loss value to be as small as possible.

Log loss can be easily extended to multiclass settings,

- Softmax classifier is basically an extension of logistic regression to multiclass.
- In Softmax classifer for C clasess we have C output neurons, each output a probability value. If z_i is the net input to i^{th} output neuron, the probability value returned by that neuron is $\frac{e^{z_i}}{\sum_{i=1}^{C} e^{z_i}}$.
- Softmax classifier minimizes the log loss for multi class (**cross entropy**), where as the logistic loss minimizes the log loss of binary class.

Let the predicted probability of an instance $x_i \in C_j$ be p_{ij}. The log loss function for multiclass can be written as Eq. 7.14, where $y_{ij} = 1$ if $x_i \in C_j$, and $y_{ij} = 0$ if $x_i \notin C_j$

$$log - loss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \log (p_{ij}) \tag{7.14}$$

- The best value of log loss is 0 and the other values can be non-zero.
- One pitfall of this log loss method is that it is difficult to interpret.

7.2 Evaluating Regression Models

Regression models predict a continouse value. The difference between the groundt truth value and predicted value is called a **residual**. In this section, we discuss various metrics that are used to assess the performace of regression models.

7.2.1 Mean Absolute Error

The **mean absolute error (MAE)** is the simplest regression error metric to understand. The MAE is calculated using the Eq. 7.15

- We calculate the residual for every data point in the test dataset, taking only the absolute values, so that negative and positive residuals do not cancel out.
- We then take the average of all these residuals.
- As we are taking the absolute values of residual, we do not consider the model is overshooting or undershooting the ground truth.
- Here, larger residuals will contribute linearly to the overall error.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{7.15}$$

7.2.2 Mean Squared Error

The **mean squared error (MSE)** squares the residuals before summing them all. MSE is calculated using the Eq. 7.16

- The error grows quadratically in MSE and outliers contribute to more errors. It penalizes the higher residuals more than lower residuals.
- Root Mean Squared Error (RMSE) is the square root of the MSE.
  - Because the MSE is squared, its units do not match that of the original output.
  - We will often use RMSE to convert the error metric back into similar units, making interpretation easier.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{7.16}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{7.17}$$

7.2.3 Mean Absolute Percentage Error

The **mean absolute percentage error (MAPE)** is the percentage equivalent of MAE. MAPE value is calculated using Eq. 7.18.

$$MAPE = \frac{100}{N}\sum_{i=1}^{N}\Big|\frac{y_i - \hat{y}_i}{y_i}\Big| \tag{7.18}$$

- Percentage values are easier to interpret than absolute values.
- MAPE is undefined for data points where the ground truth value is 0.
- MAPE can grow unexpectedly large if the ground truth values are exceptionally small.

7.2.4 R^2 - Coefficient of determination

R^2 coefficient method measures how well a model performs when compared with a simple mean model. i.e the model that predicts mean value of the groud truth for an test instance.

We compute the mean of sum of squared residuals from mean value of ground truth - SS_{mean} and the mean of sum of squared residuals from the predicted values. $SS_{residue}$ and then R^2 coefficient is computed using these two metrics as given in Eq. 7.21.

$$\text{SS\_total} = \sum_{i=1}^{N}(y_i - \bar{y_i})^2 \tag{7.19}$$

$$\text{SS\_residue} = \sum_{i=1}^{N}(y_i - \hat{y_i})^2 \tag{7.20}$$

$$R^2 = \Big(1 - \frac{\text{SS\_residue}}{\text{SS\_total}}\Big) \tag{7.21}$$

The R^2 coefficient can be interpretted as given on Table 7.3. This method is not robust against outliers, but we can use median absolute deviation to resolve this problem.

7.3 Model Selection

Model selection refers to choosing best performing model from a list of models. In this chapter, we discuss about various methods that helps to choose the right model.

Table 7.3: Interprettaion of R^2 value

R^2 **value**	**Interpretation**
$R^2 = 1$	This model is the best, because $SS\_residue = 0$ and all response values are matching perfectly
$R^2 = 0$	This model is same as simple mean model
$0 < R^2 < 1$	This model is better than simple mean model
$R^2 < 0$	This model is worse than simple mean model

7.3.1 Bias Variance Trade off

Imagine we are given a dataset with weight as predictor variable and height as dependent variable for a gang of people. We want a machine learning model to capture the relationship between input and output and predict height given a weigt. There is generally a positive linear relationship between weight & height. At one point the weigt increases but the height and people become obese. So the line will turn into a curve for weight values beyond that point. We want our model to understand this relationship and fit the data.

- If we fit this dataset with a line (linear models), we may observe high error for training samples. So the inability of a model to properly capture the relationship between input and output is knows as the **Bias**. Lower the bias better is the model in capturing the input output relationship.
- If we fit this dataset with non linear models (like decision trees), we may observe the model fits the training data well. i.e low bias. But if we are observing this model has high error on test dataset (unseen data), we conclude the model doesn't generalize well. So the inability of model to generalize on any unseen data is termed as **Variance**. Lower is the variance better is the model in generalization.
- Its hard for any model to pick the best bias and best variance scores. Only thing we can do is to find a *sweet spot* between bias and variance. So we say there is always a *trade-off* between bias and variance.
- There are methods that helps us to find the sweet spot or atleast closer to the sweet-spot.
  - Hyper parameters tuning (Grid search, Random Search etc.)
  - Regularization (L_1 regularization, L_2 regularization etc).
  - Bagging methods (Random Forest)
  - Boosting methods (Adaboost, Xgboost etc.)

- Bias is often the result of low model complexity and variance is a result of high model complexity. Bias leads to **underfit** of model and variance leads to **overfit** of model.

Underfit of model & Overfit of model is shown in Fig. 7.6.

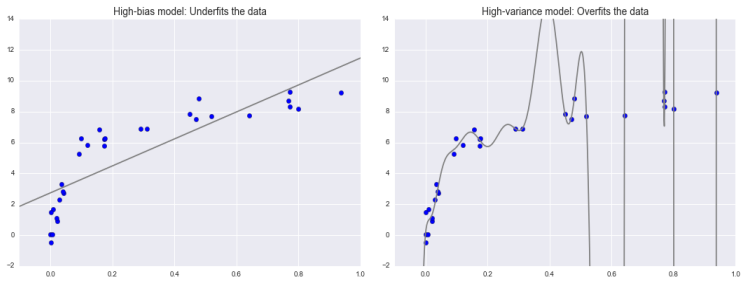
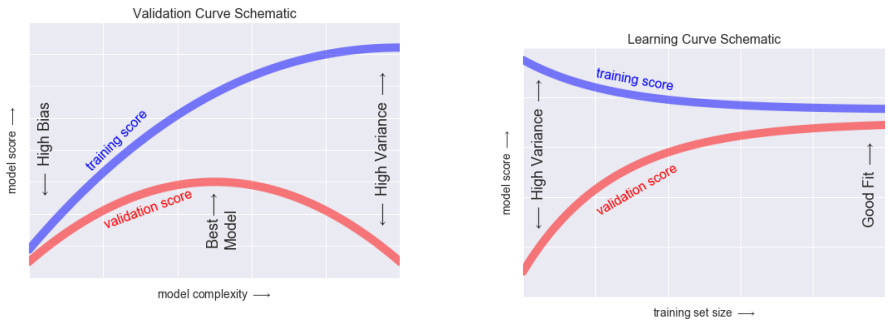


Figure 7.6: Underfit model & Overfit model

If we plot training score v/s validation score, we tyically expect to see a curve called **validation curve** as shown in Fig. 7.7a and if we plot training/validation score against training dataset size, we get a curve called **learning curve** as shown in Fig. 7.7b.



(a) Validation curve (b) Learning curve

Figure 7.7: Validation curve & Learning curve

- Increasing the training data generally improves the model performance to some extend. This is because as we increase N, the rate at which test error decreases is higher than the rate at which training error increases.
- Typically larger dataset can support a much more complicated model without being easily get overfit.
- If we fix the model complexity, typically smaller datasets leads to overfit and larger datasets leads to underfit.

- The notable feature of the learning curve is the convergence to a particular score as the number of training samples grows. Beyond this point,
  - Adding more training data will not help in improving generalization ability of model.
  - The only way to increase model performance is to use another model (usually more complex model).

The error of the model prediction on unseen data is known as the **Generalization error** (G_e). G_e has 3 componensts as shown in Eq. 7.22 where B_e^2 is the error caused by bias, V_e is the error caused by variance and I_e is the irreversible error - i.e the error which cannot be corrected. I_e has occured due to some reasons which are beyond the scope. For eg: While testing a new drug on a set of patients some patients already may have better health conditions and some may have pre existing conditions. So this type of underlying conditions can result into irreversible error in the model.

$$G_e = B_e^2 + V_e + I_e \tag{7.22}$$

- There is quadratic relationship between bias and generalization error, so to reduce it we need to focus more on bias than variance.
- V_e : Another way of looking at variance of a model is, If the model score varies much with a small change in training data, we say the model has high variance.

7.3.2 Holdout Method and Random Subsampling

We hold back some subset of the data from training dataset of the model, and then use this holdout set to check the model performance. Fig. 7.8 illustrates the hold out method.

- In this method, the given data are randomly partitioned into two independent sets, a training set (typically $\frac{2}{3}$ of data) and a test set (typically $\frac{1}{3}$ of data).
- One disadvantage of using a holdout set for model validation is that we have lost a portion of our data to the model training.
- **Random subsampling** is a variation of the holdout method in which the holdout method is repeated k times. The overall accuracy estimate as the average value of accuaracies.

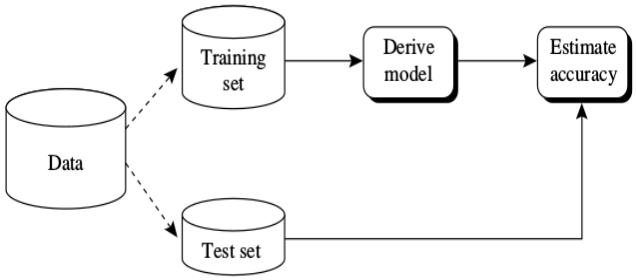


Figure 7.8: Holdout method

7.3.3 Cross-Validation

In k-fold cross-validation, the initial data are randomly partitioned into k mutually exclusive subsets or folds, $D_1, D_2, \ldots, D_k$ each of approximately equal size. k-fold cross validation is shown in Fig. 7.9.

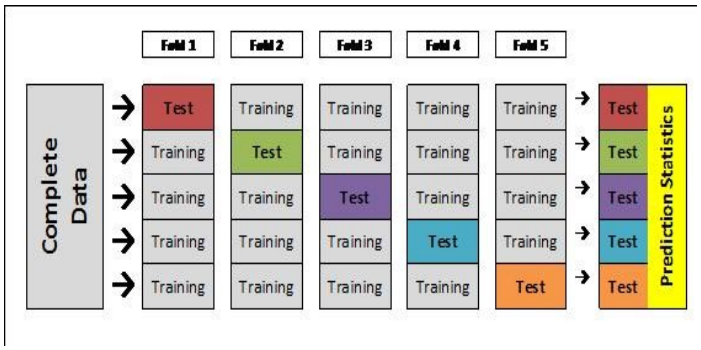


Figure 7.9: Cross Validation Method

- Training and testing is performed k times. Unlike the holdout and random subsampling methods, here each sample is used the same number of times for training and only once for testing.
- Its known as cross-validation method because, each subset of the data is used both as a training set and as a validation set. Average of the scores from multiple rounds are used to assess the model performance.
- **Leave-one-out** is a special case of k-fold cross-validation where k is set to the number of tuples in the dataset. That is, only one sample is "left out" at a time for the test set and all other samples used for training.
- In **stratified cross-validation**, the folds are stratified so that the

class distribution of the tuples in each fold is approximately the same as that of the initial data.

- In general, **stratified 10-fold cross-validation** is recommended for estimating scores (even if computation power allows using more folds) due to its relatively low bias and variance.

7.3.4 Bootstraping

Bootstrapping is a data sampling procedure that resamples a single dataset to create many simulated samples. This process allows us to,

- calculate standard errors and other test statistics like mean, median, mode, standard deviation etc.
- construct confidence intervals
- do hypothesis testing

Bootstraping methods are alternative approaches to traditional hypothesis testing and are notable for being easier to understand and valid for more test statistics.

Suppose we have a new drug developed and wanted to test how effective is the drug in curing a disease. Assume we have given the dose to 100 people and 60% of them felt that it has helped them to improve their health conditions but for 40% it didn't work. We want to check the 60% of the cases where improvement is noted is by chance or not. One way of testing this is by replicating the experiment multiple times and record the values and compute the test statistics and build a histogram. From the histogram we can evaluate the mean of the distribution and see the drug has worked or not. But this method is time consuming and costly. Bootstraping is a way to do this test quickly and efficiently and at the same time it simulates the test results like replicating the experiment multiple times.

- Each time a tuple is selected, it is equally likely to be selected again and re-added to the training set (uniform sampling with replacement)
- If the orignal data set size is N, each bootstraped sample is of size N - this is possible only with samplig with replacement method.
- We use bootstrapping method to partiiton the dataset into training set and test set. The data tuples that did not make it into the training set end up forming the test set.
- Each tuple has a probability of $\frac{1}{N}$ of being selected and $(1 - \frac{1}{N})$ not being choosen.
- We have to select N tuples, so the probability that a tuple will not be chosen during this whole sampling process is, $(1 - \frac{1}{N})^N$.

7.3.5 Hyper Parameter Search

In case of logistic regression we have one parameter $\lambda \in R$, since the λ value is real number we have an infinite search space. There is a great difficulty in searching continuos space for the optimum λ value as opposed to searching in discrete space as in KNN hyperparameter values.

- **Grid search** is a technique we can use to search for λ value. Here we start searching for the λ value in a range like $[0.001, 0.01, 0.1, 1.0, 10.0]$ etc.
  - We need to train the model with each of the λ value and cross validation error needs to be estimated for each.
  - In this method, as the number of hyper parameter increases the number of times we need to train the model increases exponentially.
  - This is almost like a bruteforce method.
  - In case of elastic net model, we need to search for two λ_1, λ_2 values using grid search.
- **Random search** is another method which is equally good as that of grid search. As the name suggests it does a random search.
  - As the number of hyper parameter to be searched increases the performace of random search may be better than grid search in terms of time.

We can get the feature importance from the weight vector.

- For feature importance in logistic regression we assume that features are independent. Since logistic regression is similar to Gaussian Naive Bayes in terms of probabilistic interprettation, the assumption of feature independence is mandatory.
- If the absolute value w_j in weight vector is large featue j is very important.

7.4 Data Science Life Cycle

The various stages in data science life cycle are :

- **Understad business requirement :** Understand problem statement or customer requirements.
- **Data aqusition :** (ETL , SQL or data warehouse, Hadoop spark, log files etc)

- **Data preparation :** Data cleaning or data preprocessing (Featurization, Dimensionality reduction)
- **Exploratory data analysis :** Plots, statistics etc.
- **Modelling evaluation and interpretation :** Classification, Regression, Clustering etc.
- **Communicate results :** Communicate with customer etc. Result charts , plots etc.
- **Deployement of Models :** Deploy model into production.
- **Real world testing:** like AB Testig, Canary testing etc.
- **Customer Buy in**
- **Operations / Maintainance :** Retraining models, handle failures etc.
- **Optimization :** Code optimization stage, handle more data, improve accuracy , business scaling etc.

8

K Nearest Neighbor

Whenever a new situation occurs, we scans through all past experiences that we had and looks up the closest experiences to the new situation. **K-nearest neighbor (KNN)** is an algorithm inspired from this type of reasoning. Those closest experiences are what we call the k-nearest neighbors. The idea of nearest neighbor search find applications in areas like:

- Pattern recognition - in particular for optical character recognition - identify the most similar character to a given handwritten character.
- Spell checking - suggesting correct spelling. Find the most similar word to the given word.
- Plagiarism detection - Identify the document which is most similar to the given document.

Suppose we want to predict the label for a data point x_q (query point), using KNN algorithm. There are three major steps in predicting the label for the query point. Fig. 8.1 shows the idea for predicting label for classification problem. Same idea can be used for regression problem with a slight change in the prediction step.

1. Find k nearest neighbours $(x_1, x_2, \ldots, , x_k)$ from the data set using a suitable similarity measure. Distances like Euclidean, Manhattan, Cosine etc. We need to select the similarity metric carefully based on the problem at hand. The choice of this metric can significantly impact the performance of the model.
2. Find the labels of these points from the training data set.
3. Using the neighborhood labels, predict label for x_q based on :
    - majority vote for classification problem
    - averaging for regression problems

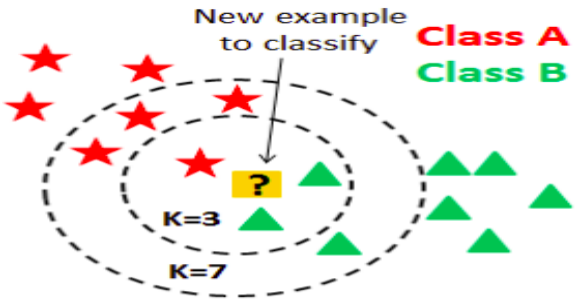
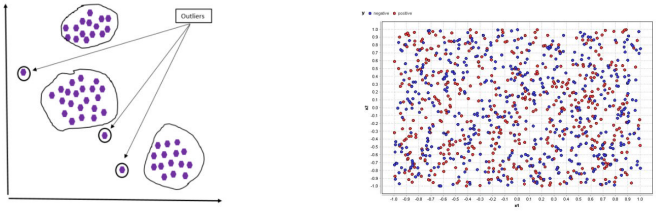


Figure 8.1: KNN for Classification

- KNN algorithm can be naturally extended to multi-class classification problem due to the majority voting step.
- The drawback of KNN is that it requires excessive amount of memory for large train data set, since we need to retain the train data set for prediction. Due to this KNN can not be used in low latency applications and large data sets.
- KNN is called **lazy learner** as there is no effective training here in contrast to **eager learners** which will construct a generalized model by effective learning before performing any prediction on query point.
- KNN neighborhood is formed using the similarity or distance between data points. So if we are given a similarity matrix or dissimilarity matrix for training data points, we can easily compute the nearest neighbours of a data point.

Two failiure reasons for KNN are outliers and equally mixed up data as dipicted in Fig. 8.2.



(a) Outliers (b) Equally mixed up data

Figure 8.2: KNN Failiure Cases

The time complexity and space complexity of the algorithms is $\mathcal{O}(Nd)$, where N is the number of data points in train dataset and d is the number of features. The time it takes to estimate the optimal value of k in KNN, increases t times if we use t-fold cross validation.

Like other machine learning algorithm the hyper parameter of KNN needed to be tuned using cross validation method. $k \in [1, N]$ is the only one hyperparameter for KNN. Best value of k lies in $(1, N)$. The k value significantly impact the model performance as follows.

- $k = 1$: It ensures that no error in training data. This shows the *overfit case.*
- $k = N$: Since we are using majority vote/averaging, every data point will be classified as the majority class and average value of labels for regression problems. This shows *underfit case*

We can improve the KNN method by assigning a weightage to each of the neighbours (**Weighted KNN**). The weight to a neighbour can be assigned as the reciprocal of distance from the query point.

- For classification problem instead of using majority vote, compute the weightage of each class and then assign a class label which is having the highest weight.
- For regression problems, weighted average of labels could be used.

Similarity computation is a costlier step in finding k- nearest neighbors. Hash table based methods can be used for reducing the time complexity for nearest neighbor search. A randomized algorithm - Locality Sensitive Hashing (LSH) is a method based on hashing for finding the nearest neighbors.

Some pros and cons of this algorithm are :

- Test stage is slow as we need to evaluate the similarity with every data point in the training set and this time is independent of the choice of k
- Scaling is mandatory as we use similarity metrics here.
- Imbalance dataset can easily impact the performance of model
- It cannot deal with missing values. As we are using the similarity metric, we cannot substitute unknown tag for missing values.
- Its not suited for high dimensional dataset due to similarity metrics being used.
- Its not robust against irrelevant features instead all features are given same importance until unless we plug in a custom similarity metric.
- The model performance get negatively impacted if there is redundancy in the dataset.

9

Regression

Say you have a business for selling products in retail vertical and you want to know how much sales we can expect based on the customers past purchasing behaviours and amount spent on marketting (advertising) etc. Your sales forecast is the foundation of the financial story that you are creating for your business. But beyond just setting the stage for a complete financial forecast, your sales forecast is really all about setting goals for your company. You're looking to answer questions like:

- What do you hope to achieve in the next month? Year? 5-years?
- How many customers do you hope to have next month and next year?
- How much will each customer hopefully spend with your company?

Your sales forecast will help you answer all of these questions and potentially any others that involve the future of your business.

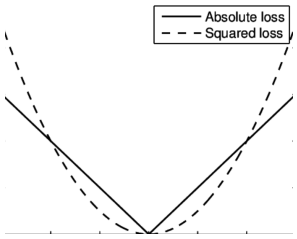
Figure 9.1: Sales Forecast

We want to build a machine learning model that learns from the data and predict the results as accurately as possible. The amount of sales is a *continuos value* that we want to predict and this is a **linear regression** problem, a supervised learning problem.

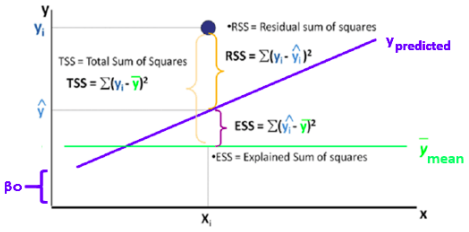
9.1 Linear Regression

Predicting the sales requires us to fit the regression model to given data. Linear regression is one of the popular class of regression algorithm under **Generalized Linear Models (GLM)**. In linear regression we *fit the best hyperplane* that describe the data.

Assume we want to predict the sales based on the marketting cost (for advertising) we spent. On X axis we have the marketing cost and Y axis denote the sales. Fig. 9.2b shows the best fit line (blue) and mean value of ground truth (green line) denoted as $(\bar{y})$. The difference between the predicted value $(\hat{y}_i)$ and the ground truth (y_i) is called a **residual**. Residual are squared while calculating errors, this is done to avoid the cancelling out of positive and negative errors, penalize large errors more. Also the squared function is differentiable everywhere and easier to work with graidnet descend methods.



(a) Squared loss ($Y = X^2$)



(b) Squared Errors

Figure 9.2: Squared Loss & Linear Regression

- Residual denote the error in prediction and we want to minimize it. So fitting a line in linear regression is also known as **fit of least squares** or **ordinary least squares (OLS)**.

- If we have only one variable to predict the depended variable then its called the **simple regression**, where as the **multiple regression** has more than one predictor variables.

- We can compute the variation of depended variable from its mean (variance) and variation of depended variable from the independent varaiables. These two variance can be combined to R^2 **coefficient**.

R^2 coefficient explains the varaiance of sales while taking into account the predictor variables like marketing cost, number of customers etc. It basically explains the reduction in variance of depended variable while taking into consideration of the independent variables. its calulated as follows,

$$R^2 = \frac{\Sigma(y_i - \bar{y_i})^2 - \Sigma(y_i - \hat{y_i})^2}{\Sigma(y_i - \bar{y_i})^2} \tag{9.1}$$

- $R^2 = 0$ represents the worst fit as we have 0% reduction in variance with reference to the mean value of depended value, where as $R^2 = 1.0$ denote the best fit model which shows 100% reduction in variance or zero error.
- R^2 can also be interpretted as ratio of variation of model by taking the independent variables into account to the variation of model by considering only the depended variable.

The concept of R^2 holds true in high dimensions where we have many predictor variables. But having many independent variables which have random chances on improving the R^2 value may lead to mis interprettation of the performance of model. How can we be sure that the R^2 is reliable when such random chance predictor variables are present in the data? Solution is **p-values** i.e we need to statistically determine if R^2 value is significant. The p value comes from F-distribution.

- F-distribution is the ratio of variation of model by taking the independent variables into account to the variation of model not explained by independent variables (residuals).

The F-value is calculated using Eq. 9.2 where N is the number of data points, p_f, p_m are the number of parameters of fit model and mean model respectively. $p_f - p_m$ denote the number of additional variables available for prediction when compared with mean model. For fitting a line we need minimum 2 points, for plane we need minimum 3 points and so on, so $N - p_f$ denote the excess data points available in the data set.

$$F = \frac{\frac{\Sigma(y_i - \bar{y_i})^2 - \Sigma(y_i - \hat{y_i})^2}{(p_f - p_m)}}{\frac{\Sigma(y_i - \hat{y_i})^2}{(N - p_f)}} \tag{9.2}$$

If the model fit is good the numerator in the equation of F will be high and denominator will be too small. This results in a large value of F. But how do we turn this into a p value? This is done by generating many random data points and estimating the F value on each of the data sets and recording the F values on a histogram. Then with the original data set compute one F value and add that to the histogram. p value is the number of points in the histogram that is greater than or equal to the F value obtained on original dataset divided by the total number of points.

Computing the p values using histogram is computationally expensive and we do not generally follow that in practice. Instead we use F-distribution curve and estimate the p value from that curve.

- For a good fit regression, the R^2 value should be high and p value for the R^2 should be low.

The slope of hyperplane (weights) decides the bias variance trade off in regression model. If the slope is high (maginitude of wieight vector is high) this makes the model more sensitive to the weights and leads to overfitting. Larger the maginitude higher the change in depended varaibles. So how do we prevent the weights reaching high values? The solution is **regularization** which adds a **regularization penalty** in addition to the least square loss or residual loss. Regularization *shrinks the weights* of model and helps to reduce the variance at the cost of slight increase in bias.

- Adding regularization in the model helps to give lower weightages to (zero out) all spurious predictor varaibels like outcome of flipping a coin and other unrelated predictor variables. This prevents overfitting.

Adding the regularization penalty force the objective function to balance between the penalty and the squared loss. This may slightly increase the least squared loss but shrinks the weights. Now the regularization penalty that needs to be controlled to find the right balance, this is done by adding a hyperparameter $\lambda \geq 0$. Hyperparameter λ is tuned using cross validation method.

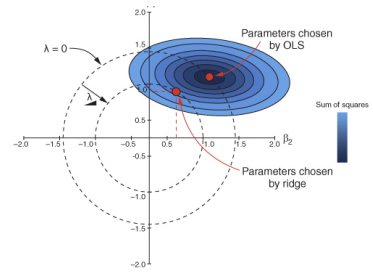
Depending on how we compute the maginitude of weight vector, there are three types of regularizations **Ridge Regression (L_2 norm)**, **Lasso Regression (L_1 norm)** and **Elastic Net (L_1&L_2 norm)**. Let N be the number of data points and W represents the d dimensional weight vector and let w_j be the weight of j^{th} predictor variable and w_0 be the intercept. The objective function minimizing the squared loss for linear regression with different regularization is given below where $\alpha \in [0, 1]$ denote the fraction of L_1, L_2 regularizations.

$$\hat{W}, \hat{w_0} = \underset{W,w_0}{\arg\min} \sum_{i=1}^{N} (y_i - \hat{y_i})^2 + \lambda ||W||_1^1 \tag{9.3}$$

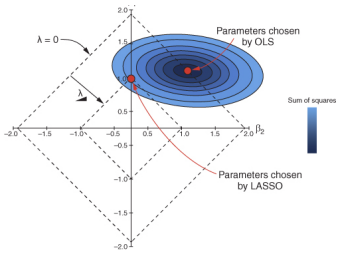
$$\hat{W}, \hat{w_0} = \underset{W,w_0}{\arg\min} \sum_{i=1}^{N} (y_i - \hat{y_i})^2 + \lambda ||W||_2^2 \tag{9.4}$$

$$\hat{W}, \hat{w_0} = \underset{W,w_0}{\arg\min} \sum_{i=1}^{N} (y_i - \hat{y_i})^2 + \alpha ||W||_1^1 + (1-\alpha)||W||_2^2 \tag{9.5}$$

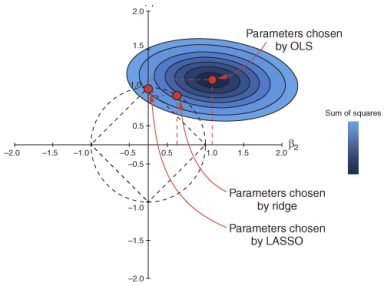
Lets understand the difference of these regularization methods in detail. To explain the concept, lets assume the model has two weight parameters and data points are two dimensional. Fig. 9.3 shows the difference in parameter selection by Ridge, Lasso and Elastic Net regularization penalties. X, Y axes represent the parameter value and contours represent the squared loss. The solid concentric circles represent the sum of squares of errors (Oridnary Least Squares), dashed circles represents the L_2 norm multiplied by λ, dashed diamond represents the L_1 norm multiplied by λ and yellow coloured region represents the combination of L_1 & L_2 with α and $1 - \alpha$ proportions respectively.



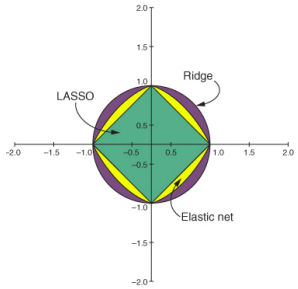
(a) Ridge Regression (L_2)



(b) Lasso Regression (L_1)



(c) L_1 vs L_2 Penalty



(d) Elastic Net (L_2 & L_1)

Figure 9.3: Types of Regularization Penalties

- L_1 regularization derivative is 1 or -1 , so it pushes the weights of unimportant features to zero quicky and zero out. So this has the benefit of feature selection to reduce dimesnions. L_2 regularization on the other hand has the derivative of $2W$, as the weight approaches 0, the gradient aproaches zero which results in stand still in weight update.
- If we have some prior belief or domain knowledge that some featues are unimportant and also our data set has high dimensions its better to use L_1 regularizer, where as if we are have prior belief that all featues do contribute in prediction we can use L_2 based regularization. Elastic Net is a balance between L_1&L_2 and can be swithced to either of the versions by setting $\alpha = 0$ or $\alpha = 1$.

Here we derived the expression from the geometric perspective, we can also derive it from the probabilistic perspective.

Simple and multiple linear regression models are used to model the relationship between explanatory variables and a quantitative response variable. How do we use this model to predict the discrete response variable . i.e How do we turn this model into a classification model? **Logistic Regression !!** In spite of its name, *logistic regression is a model for classification.*

9.2 Logistic Regression

In general we call the two outcomes of the response variable "success" (denoted as 1) and "failiure" (denoted as 0). Then the mean value of this response variable denote the probability of success event denoted as p and is the proportion of success in the dataset. $p-1$ is the probability of failiure. Let our dataset has N independent observations, then this is a **binomial distribution** setting. In addition to this distribution the probability value p depends on one or more explanatory variables. Just as in multiple linear regression, the explanatory variables can be either categorical or quantitative. Logistic regression is a statistical method for describing these kinds of relationships.

Lets take a group of 250 customers out of which 210 purchased a product. The proportion of people who purchased the product is $\frac{210}{250} = 0.84$. Logistic regression works on **odds** rather than proportions. Odds are the ratio of proportion of two events that are complements to each other.

$$\text{odds} = \frac{\frac{p}{N}}{\frac{1-p}{N}} = \frac{p}{1-p} \tag{9.6}$$

Based on the above idea, odds of the customers who purchase the data is $\frac{0.84}{0.16} = 5.25$. Usually when people talk about odds they round this to integers as 5 or $\frac{5}{1}$ i.e odds are 5 to 1 that customers purchase a product. But in logistic regression we use the odds value as it is. In a similar way, we could describe the odds that a customer would not purchase the product as 1 to 5.

Lets assume 61.08% of women and 43.98% men purchased the product, what are the odds for purchase of the product by men and women?

$$\text{odds}_{\text{men}} = \frac{0.4398}{1-0.4398} = 0.7851$$
$$\text{odds}_{\text{women}} = \frac{0.6108}{1-0.6108} = 1.5694$$

Lets try to model the probability of purchase of product p using just one predictor variable gender encoded as (men=1, women=0). Like linear

regression model lets combine the predictor variable with weights and try to model the problem to predict the probability as shown in Fig. 9.4.

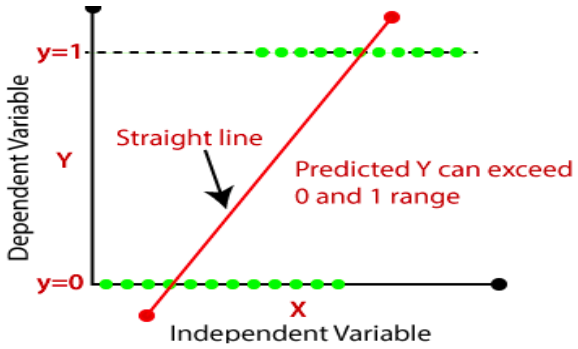


Figure 9.4: Modelling Logistic Regression

Let $X_1 \in \{0, 1\}$ denote the gender and we model $p = w_1 X_1 + w_0$. This model has limiatations as right hand side of equation can have values in $(-\infty, \infty)$ but the left hand side has values in range $[0, 1]$. So we need a better modelling choice as follows

- We need to extend the range of left hand side to $(-\infty, \infty)$ and at the same time the left hand side term should contains the probability p in it. We can do it by the transformations on left hand term, by first taking the odds ratio and then applying natural log to it. Applying log will extend the odds to $(-\infty, \infty)$, so it creates symmetry for odds value between $[0.0 - 1.0]$ & $[1.0, \infty)$. This log based odds transformation is called **Logistic transformation** - i.e log of odds function **logit function**.

Now the probability of a person purchases the product can be written as in Eq. 9.7 and is called **simple logistic regression** and the idea can be extended to more than one predictor variables (eg: gender, income, location etc.) called **multiple logistic regression** and can be written as Eq. 9.8.

$$\ln\left(\frac{p}{1-p}\right) = w_1 X_1 + w_0 \tag{9.7}$$

$$\ln\left(\frac{p}{1-p}\right) = W^T X + w_0 \tag{9.8}$$

We discussed a case where we have only one predictor variable gender which is an indicator variable. In such case its easy to find the regression coefficients by substituting the gender value and equating to log odds. If we have numeric predictor variables or more than one predictor variables computing the coefficients are done by gradient descend method.

The above equations gives only the logit value, how do we get the probability of a user will convert or not? Its done by applying inverse of logit function called **Sigmoid function** $\sigma(Z)$ that output probability as shown in Fig. 9.5 and formula given in Eq. 9.9.

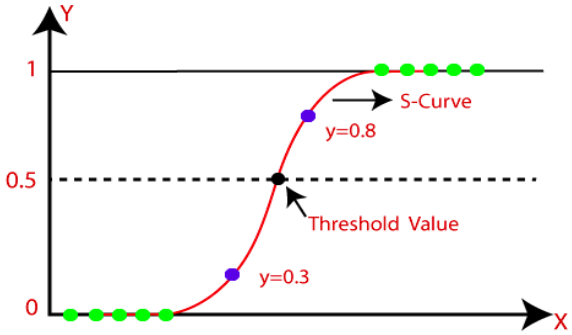


Figure 9.5: Sigmoid Function

$$\sigma(Z) = \frac{1}{1 + e^{-Z}} \tag{9.9}$$

There are couple of differences between linear regression fitting and logistic regression fitting.

- Logistic regression fits a transformed (logit) version of output varaiable as opposed to original output variable as in regression. Logistic regression tries to fit a sigmoid curve where as in linear regression we fit the hyperplane.
- We need to do the logit transformation on the class labels too. This will map the class probability 1.0 to $+\infty$ and class probability 0.0 to $-\infty$. This creates trouble in fitting a hyperplane for logistic regression as the residuals are either $\pm\infty$. So the method of least squares doesn't work for logistic regression.

How do we fit the hyperplane then? The idea is to map the transformed space back to probability space by applying the sigmoid transformation. Now we get the probability of each data point being in positive class.

- Our objective function would be to maximize the probability of each data point belong to the correct class.

Let $P(y_i|X_i, W)$ denote the probability of data point X_i belong to +ve class, then $1 - P(y_i|X_i, W)$ denote the probability of data point belongs to -ve class. Let $y_i \in [0, 1]$ denote the ground truth probability of data point X_i. So our objective function become to maximize $L = \left(P(y_i|X_i, W)^{y_i}\right) *$

$\Big(1 - P(y_i|X_i, W)\Big)^{1-y_i}$ for data point X_i. This product (L) is known as the likelihood and hence the obective function is known as the **maximum likelihood** method. Based on this idea we can learn the weights for logistic regression hyperplane as follows,

To explain how we can derive the cost function for logistic regression, let's first define the likelihood L that we want to maximize when we build a logistic regression model, assuming that the individual samples in our dataset are independent of one another. The formula is as follows:

$$L(W) = P(y|X, W) = \prod_{i=1}^{N} P(y_i|X_i, W) = \prod_{i=1}^{N} (\sigma(Z_i))^{y_i} \Big(1 - \sigma(Z_i)\Big)^{1-y_i} \tag{9.10}$$

In practice, it is easier to maximize the (natural) log of this equation, which is called the **log-likelihood** function:

$$\ln(L(W)) = \sum_{i=1}^{N} \Big[y_i \ln\Big(\sigma(Z_i)\Big) + (1 - y_i) \ln\Big(1 - \sigma(Z_i)\Big)\Big] \tag{9.11}$$

- Firstly applying the log function avoids the numerical underflow condition.
- Secondly, we can convert the product of factors into a summation of factors, which makes it easier to obtain the derivative of this function via the addition trick. Also the addition is faster than multiplication hence speed up in execution.

Let's rewrite the log-likelihood as a cost function J that can be minimized using **gradient descent** as given in Eq. 9.12. This loss is known as **log loss** and is shown in Fig. 9.6.

$$J(W) = -\sum_{i=1}^{N} \Big[y_i \ln\Big(\sigma(Z_i)\Big) + (1 - y_i) \ln\Big(1 - \sigma(Z_i)\Big)\Big] \tag{9.12}$$

The gradient descend optimization makes sure that we get a good fit on the data. But how do we make sure that the model is useful? In other words, how do we compute the R^2 & p-values. We do not have residuals to compute R^2 like in linear regression, how do we get the R^2 and p-values then?. There are a number of methods suggested to compute the R^2 value and one such popular method is based on likelihood. Let N_P, N_N be the number of positive & neagative obseravations in the dataset. The R^2 value is computed as follows.

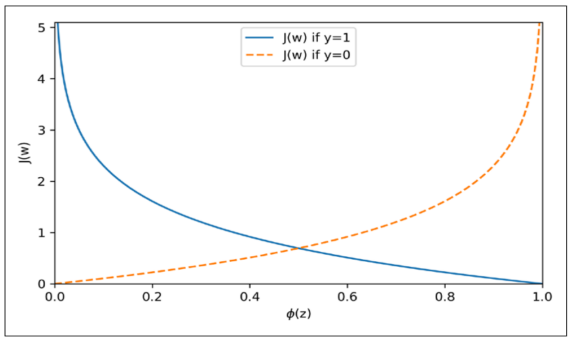


Figure 9.6: Log Loss

- Take the prior probabilities for +ve ($p^+ = \frac{N_P}{N}$) & -ve ($p^- = \frac{N_N}{N}$) clasess from the set of data points. Use these probabilities to compute the reference likelihood for the model as $L(M) = p^{+N_P} * p^{-N_N}$. $L(M)$ is analogus to mean model in linear regression.
- We can compute the likelihood of fit by taking the product of probabilities for all data points. i.e. $L(W)$
- Finally the R^2 value is computed using the formula $\frac{L(M)-L(W)}{L(M)}$. The p-value can be computed from chi-squared test and we can reject or accept the significance of R^2 value.

Now we have understood learning of weights for logistic regression model from a proabilistic perspective. Another way to understand the weight learning for logistic regression is through the geometric perspective. For explaining the geometric intuition, lets slghtly change our notation of ground truth. Let $y_i \in \{-1 \text{ (-ve class)}, 1(+\text{ve class})\}$ be the class labels. This notation is used to simplify the formula for objective function.

If datapoint X_i lies on the hyperplane then $W^T X_i = 0$. If datapoint X_i lies on the same direction as that of the unit normal vector $\hat{w}$ of the plane then then the dot product $W^T X_i$ is +ve, if X_i lies on the opposite direction as that of $\hat{w}$ then dot product $W^T X_i$ -ve. The objective function $J(W)$ of logistic regression can be defined based on this idea. The product $y_i W^T X_i$ will be:

- **Positive :** If the data point X_i is correctly classified by the model.
- **Negative :** If the data point X_i is missclassified by the model.
- **Zero :** If the datapoints X_i lies on the hyperplane.

So the objective function $J(W)$ can be written as Eq. 9.13. But this equation has high values when the model make correct prediction for outliers. This is because the magnitude of dot product is very high and then

multiplying it with correct class label will make it a huge positive number. Maximizing this quantity would be in favour of outlier data points.

$$J(W) = \max \sum_{i=1}^{N} y_i W^T X_i \tag{9.13}$$

The only parameter to learn is the weight vector W, which can be estimated using Eq. 9.14

$$\hat{W} = \arg\max_{W} \sum_{i=1}^{N} y_i W^T X_i \tag{9.14}$$

The outliers have great impact on this objective function of logistic regression. This is illustrated in Fig. 9.7.

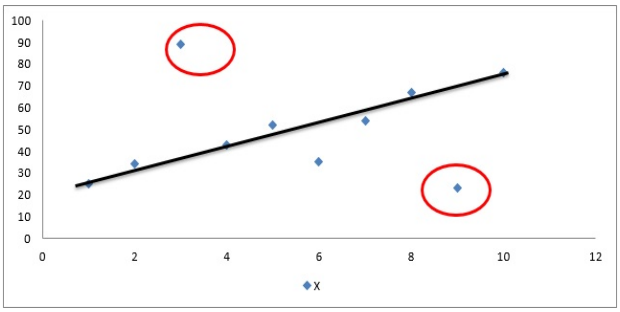


Figure 9.7: Outlier Problem in Logistic Regression

Maximization of sum of signed distances doesn't solve the outlier impact. So it is important to modify the objective function. **Squashing** is a technique which can be used to avoid this problem. Here we keep the small distances as it is and squash the large values. Many functions can be used to do squshing . eg: $\sigma(Z)$, tanh etc. We prefer sigmoid function because the output of sigmoid function $\in (0, 1)$ can be interpretted as probability.

After incorporating the squashing into Eq. 9.14, we get final formula for best $\hat{W}$ classification as Eq. 9.15 and the simplification of the formula is shown in the subsequent equations.

$$\hat{W} = \arg\max_{W} \sum_{i=1}^{N} \frac{1}{1 + e^{-y_i W^T X_i}} \tag{9.15}$$

$$
\begin{aligned}
\hat{W} &= \arg\max_W \sum_{i=1}^{N} \frac{1}{1+e^{-y_i W^T X_i}} \\
&= \arg\max_W \sum_{i=1}^{N} \ln\Big(\frac{1}{1+e^{-y_i W^T X_i}}\Big) && \text{Apply ln to simplify equation} \\
&= \arg\max_W \sum_{i=1}^{N} -\ln\Big(1+e^{-y_i W^T X_i}\Big) \\
&= \arg\min_W \sum_{i=1}^{N} \ln\Big(1+e^{-y_i W^T X_i}\Big) \\
&= \arg\min_W \sum_{i=1}^{N} \ln\Big(e^{-y_i W^T X_i}\Big) && \text{Since dropping 1 does not change the result} \\
&= \arg\min_W -\sum_{i=1}^{N} y_i W^T X_i && \text{Solve this using } \textbf{gradient descend}
\end{aligned}
\tag{9.16}
$$

Lets investigate the overfitting and underfitting in logistic regression algorithm. Recall that our objective function and net input $Z_i = W^T X_i$. When any of the $|w_i| = \infty$, the objective function can have minimum value. So this leads to overfitting but we can avoid this by adding a regularization penalty like L_1, L_2 or Elastic Net.

Let $w_1, w_2, ...w_d$ be the weights associated with each predictor variable. The probability of a datapoint belongs to a class depends on the weight vector as follows

- **Case 1:** $X_i \uparrow w_i \uparrow$ **or** $X_i \downarrow w_i \downarrow$ The probability of data point belongs to positive class increases.
- **Case 2:** $X_i \uparrow w_i \downarrow$ **or** $X_i \downarrow w_i \uparrow$ The probability of data point belongs to negative class increases.

As logistic regression uses a hyperplane it assumes the data points are linearly separable, if not then we can try feature cross or other higher dimensional projections to make the data points linearly separable. Logistic regression can be extended to multi class by One vs Rest method or using **Softmax classiifer** which is an extension of logistic regression to multi class. In contrast to Naive Bayes classifier, logistic regression directly compute the probability of data point belong to a class.

9.3 Collinearity and Multi Collinearity

Let $x_1, x_2, \ldots, x_d$ be the d features with weights $w_1, w_2, ..., w_d$. There is a problem in concluding feature x_j is important just by considering the cor-

responding absolute value of weight w_j. In real word cases the features may be collinear or multi collinear. i.e one feature could be expressed as linear combination of one or more other features. This indicates that changes in one variable are associated with shifts in another variable. The stronger the correlation, the more difficult it is to change one variable without changing another. This will make the regression coefficients (weights) become very sensitive to small changes in the features. So the final model won't be able to generalize well.

If $x_j = \alpha x_i + \beta$ where α, β are real number we say x_j, x_i are **collinear**, and if $x_j = \alpha_1 x_i + \alpha_2 x_k + \cdots + \beta$, we say $x_j, x_i, \ldots x_k$ are **multicollinear**. How do we test for collinearity? The test for collinearity among features, a **pertubation test** is often used, that has the below steps involved,

- Train the regression model on original data set and get the weight vector, pertubate dataset by adding $\epsilon \sim N(0, \sigma_s)$ ($\sigma_s \ll 1$ is very small) to columns and then train the model, so we get another weight vector.
- If the difference between respective weights in two vectors are significantly different then the features are collinear otherwise not.

If the dimension is less, we do pairwise correlation and remove one of the correlated features. If dimension is high, then we can use PCA method which reduce dimension by removing correlations. L_1, L_2 regularizers are also found to work for removing collinearity.

9.4 Time & Space complexity

Now lets look into the time & space complexity of regression models.

- Training
  - Time complexity : $O(Nd)$, we need to do multiplication d times. Space complexity : $O(d)$, we need to store only the d dimensional weight vector.
- Testing
  - Time complexity : $O(d)$, we need to do multiplication d times. Space complexity : $O(d)$, we need to store only the d dimensional weight vector.
- If d is very small, logistic regression can be used in low latency applications. if d is high use L_1 regularization with higher λ value to zero out unwanted features and reduce dimension.

10

Naive Bayes Classifier

Imagine you are receiving thousands of emails a day and some of those emails are spam. Manually going through each mail and deleting those emails are bit cumbersome and leads to poor end user experience. Your mail server wants to build a machine learning model that automatically move the spam emails to trash. **Naive Bayes Classifier** is the first machine learning algorithm build for classifying emails as spam or not.

- Naive Bayes is a **probabilistic classifier** that gives us the probability of an observation being in the class in addition to the class label of that observation.
- Naive Bayes is a **generative classifier** that will tell us the most likely class that would have generated a given observation.

Suppose we have N emails in the training data that consists of both spam and ham emails. Let S be the set of all spam emails and H be the set of all ham emails. Let $|S|$, $|H|$ denote their respective count values (frequency). The first step in building a classifier is to form features. Here we have emails (text data), so the features could be the words in the text. We can get the list of all words present in the training data and build the vocabulary.

- Naive Bayes model uses the bag of words techniques to extract the features (words in the example). So this model neither consider the order of the words nor the position where the word is present.
- Neither semantics nor the context could be learned using the bag of words representation. So this model is called a naive model.

How does the Naive Bayes algorithm classify a given observation? Lets try to build a simple model based on the Bayes's theorem in probability.

- We need an initial guess (**prior probability**) of a given observation belong to spam or ham class. This value is usually estimated from the training data. $P(S) = \frac{|S|}{N}$ & $P(H) = \frac{|H|}{N}$.

- Secondly we need the **likelihood** of each word in the observation given a class. This is the conditional probability of each word belonging to a class. We can estimate this probability by finding the total occurrences of a given word considering all observations in that class divided by the sum of frequency of all worlds in that class.

- As the last step, compute the probability of an observation belong to a class by multiplying the prior probability of that class with conditional probabilities of all words in the observation from that class. Evaluate this product of probabilities for each class and pick the class with highest probability as the class label.

For simplicity lets assume our dataset size is 100, vocabulary size is just 5 words and we want to build a classifier based on the Bayes theorem. Fig. 10.1 depicts the Naive Bayes classifier core idea.

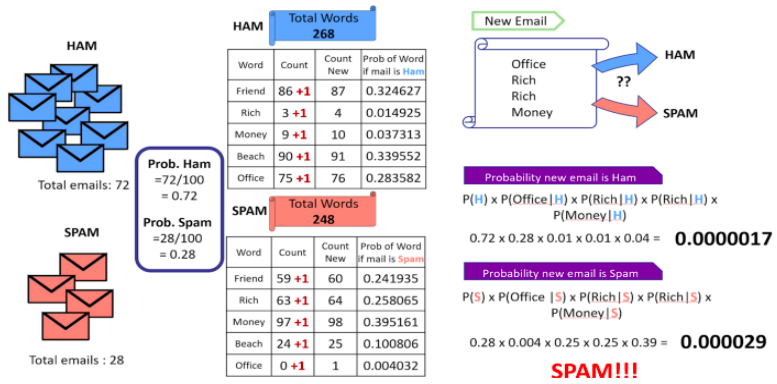


Figure 10.1: Naive Bayes Classifier

Please note that in Fig. 10.1, we have added "1" to all word counts, this is to have a non-zero value for count for all words in the vocabulary. We cannot have 0 word count or ignore a word if it is missing in one of the clasess for two reasons

- If any word is present in the test observation but not present in the vocabulary of any class, the probability of observation belong to that class will be 0.0 (due to multiplication with 0.0). So this observation cannot be assigned to that class.

- If we ignore the the word with 0 frequency then we are effectively substituting probability 1.0. This is also a wrong method.

The process of smoothing the zero probability due to the missing feature value is handled by adding a constant with the frequency. This process is know as the **additive smoothing** or **Laplace smoothing**. A more generic

form of additive smoothing for a missing feature value $f_i^{'}$ for a class y_j is determined using the below formula.

$$P(f_i^{'}|y_j) = \frac{|y_j| + \alpha}{N + K\alpha} \tag{10.1}$$

- α is the only one hyperparameter in Naive Bayes. Typically α is taken as 1. As the value of α increases the distribution becomes uniform.
- When α is very small, it causes large variance in model and results in overfitting. When α is very large, it causes large bias in model and results in underfitting.

10.1 Naive Bayes Classifiers

The basic idea of Naive Bayes classifier is to use the Bayes theorem to classify a given observation. Bayes theorem transform the probability into three other probabilities that have some useful properties as given in Eq. 10.2.

$$\overbrace{P(y_i|X)}^{\text{Posterior}} = \frac{\overbrace{P(y_i)}^{\text{prior (1)}} * \overbrace{P(X|y_i)}^{\text{likelihood (2)}}}{\underbrace{P(X)}_{\text{evidence (3)}}} \tag{10.2}$$

Naive Bayes algorithm has natural extension to multiple classes. Let $y_1, y_2, ..., y_C$ be C classes where an observation can belong to and X be the feature set with d dimensions. A generic version of Naive Bayes algorithm is given in Eq. 10.3

$$\hat{y} = \arg\max_{y_i} P(y_i|X) \tag{10.3}$$

In the above equation evidence term ($P(X)$) is independent of any class and is a constant when given an observation X and the numerator is equivalent to the joint probability model. So we can ignore denominator while computing the class probability. So the Eq. 10.2 can be rewritten as Eq 10.4

$$P(y_i|X) \propto P(y_i) * P(X|y_i) \tag{10.4}$$

As X is d dimensional vector, the **joint probability** for numerator be written as $P(X_1, X_2, X_3, \ldots, X_d|y_i)$. Using the chain rule of conditional probability we can expand the expression as

$$\begin{aligned}P(X_1, X_2, \ldots, X_d, y_i) &= P(X_1|X_2, X_3, \ldots, X_d, y_i)P(X_2|X_3, \ldots, X_d, y_i) \\ &= P(X_1|X_2, X_3, \ldots, X_d, y_i)P(X_2|X_3, \ldots, X_d, y_i)P(X_3|X_4 \ldots, X_d, y_i) \\ &= P(X_1|X_2, X_3, \ldots, X_d, y_i)P(X_2|X_3, \ldots, X_d, y_i) \ldots P(X_d|y_i)\end{aligned}$$

The above expression is very difficult to evaluate but if we make an assumption (Naive Bayes Assumption) that the features are independent, we can further simplify the expression. The idea is to use conditional independence. If we have three random variables A, B, C such that $P(A|B, C) = P(A|C)$, we call this as **conditional independence**. Now the above expression can be simplified to

$$P(X_1, X_2, X_3, \ldots, X_d|y_i) \propto P(X_1|y_i)P(X_2|y_i) \ldots P(X_d|y_i) \tag{10.5}$$

Using the above simplified formula for Naive Bayes rule, the final formula can be written as,

$$P(y_i|X) \propto P(y_i) \prod_{j=1}^{d} P(X_j|y_i)$$

We can build a classifier by applying the formula to each class and picking the class which has the highest probability. This method is called **maximum aposterior method**.

$$\hat{y} = \max_{i=1,2,\ldots,C} P(y_i) \prod_{j=1}^{d} P(X_j|y_i) \tag{10.6}$$

As we are multiplying many probability values, there is a chance numeric underflow. To avoid the numeric underflow and make the algorithm run faster we can work in log space. So we can rewrite the equation in log space as given in Eq. 10.7

$$\hat{y} = \max_{i=1,2,\ldots,C} \log P(y_i) + \sum_{j=1}^{d} \log P(X_j|y_i) \tag{10.7}$$

10.2 Extension of Naive Bayes Classifier to Numeric Features

So far we have discussed about **multinomial Naive Bayes classification** where the features are categorical type (words in text document). How do we use Naive Bayes classifier for dataset with numeric features. We can

extend the classic Naive Bayes model to deal with the numeric features as explained below.

To compute the posterior probability we need to estimate the likelihoods. If we are able to get the likelihood for numeric features then the posterior can be estimated easily. We can do this by fitting each numeric feature to a probability distribution (i.e getting parametes like μ, σ etc.).

- Post fitting to a distribution, when an observation is given we can plug its value to probability distribution and get the likelihood values. This value can be used in the posterior calculation formula.
- Two popular extension of Naive Bayes to numeric features are **Gaussian naive Bayes** for normal distribution and **Bernouli naive Bayes** for binary features.

10.3 Feature Importance, Pros & Cons

The feature importance of Naive Bayes classifier can be directly computed from model since we have the likelihood values for all the features. We can get the class wise important features by choosing the top likelihood probabilities of each feature. Model predicted a class label because it found high likelihood for some features, in this manner we can interpret the model.

Pros	**Cons**
Naive Bayes is extensively used in text data since it is of high dimensions and Naive Bayes can handle high dimensions.	Since we use the prior term in the equation the posterior probability tends to be very small for minor class. Data imbalance create trouble.
The time complexity of Naive Bayes algorithm is $O(N * d * C)$ & space completely is $O(d * C)$. So it can be used in low latency applications.	The α value effects minority class more compared to majority class in case of an imbalanced data set.
Laplace smoothing over powers the probability of outlier points.	

Table 10.1: Pros & Cons of Naive Bayes Classifier

11

Support Vector Machine

Let's begin by a motivating example for **Support Vector Machines (SVM)**. Suppose we are building a model to predict the car is expensive or not based on the features of the car. Assume our datset consists of 2 dimensions and the expensive car data points are represented as red and the cheap one are represented as the green one.

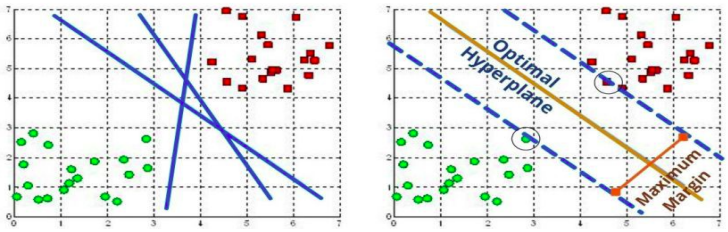


Figure 11.1: Hyperplane maximizing the margin

There are infinitly many hyperplanes to separate red data points from greens. How do we choose *optimal hyperpalne* to classify the data points then? To generalize well on unseen data, we have to choose one **hyperplane that maximizes the distance** between closest pair of data points from each class. SVM is based on this and the idea is illustrated in Fig. 11.1.

- The shortest distance between the closest separating hyperplanes is called the **Margin**. So SVM problem can be formulated as the problem of **maximizing the margin**.
- The optimal hyperplane is the one that pasess thorugh the mid point of boundary hyperplanes. The middle hyperplane is expected to generalize well on unseen data.
- Given an optimal margin linear separator, the data points through which the maximum margin hyperplanes pasess throguh are called the **support vectors**.

Maximizing the margin can also be interpretted as **convex hull problem** where we need to construct two convex hulls one for each of the class and find the shortest distance between convex hulls. This is shown in Fig. 11.2.

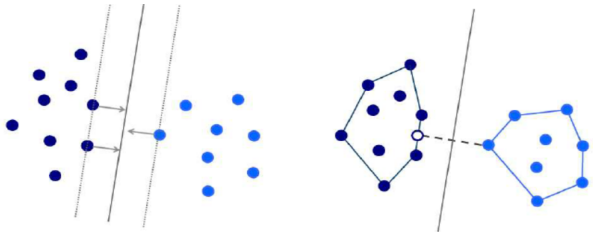


Figure 11.2: Maximizing Margin using Convex Hull

11.1 SVM Classification

We start by exploring the SVM for binary classification problem. Let our dataset is d dimensional and contains N data points. Assume the data points are linear separable (free of noise). Let $+1$ denote the class label for positve class and -1 denote the class label for negative class.

Hyperplane can be represented as a function for classifying the data points as $F(X) = W^T X + b = 0$. First, to correctly classify the training set, $F(.)$ must return positive numbers for positive data points and negative numbers for negative data points. If there exists such a linear function F that satisfies this conditions then the data points are said to be **linearly separable**. For every data point X_i we want,

$$WX_i + b > 0 \qquad \text{if } y_i = +1 \tag{11.1}$$

$$WX_i + b < 0 \qquad \text{if } y_i = -1 \tag{11.2}$$

Both conditions can be summarized into:

$$y_i(WX_i + b) > 0 \qquad \forall (X_i, y_i) \in D \tag{11.3}$$

If there exist W & b that satisfy Eq.11.3, they can always be rescaled to Eq. 11.4. The Fig. 11.3 shows maximizing the margin of SVM for 2D datapoints.

$$y_i(WX_i + b) \geq 1 \qquad \forall (X_i, y_i) \in D \tag{11.4}$$

The margin of hyperplane ($F(.)$) implies the error in classifier i.e larger the margin, lower is the classification error. Intuitively, the classifier that contains hyperplane with a small margin are more susceptible to model overfitting and tend to classify with weak confidence on unseen data.

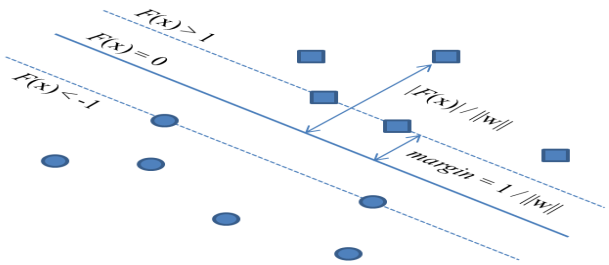


Figure 11.3: The hyperplane maximizing the margin in a 2D space

- When X_i are the closest vectors to hyperplane, $F(X_i)$ will be ± 1, and X_is are support vectors.

The distance from the hyperplane to a vector X_i is formulated as $\frac{F(X_i)}{||W||}$, Thus, the margin becomes:

$$margin = \frac{1}{||W||} \tag{11.5}$$

Maximizing the margin $\frac{1}{||W||}$ is equivallent to minimizing $||W||$. Thus the training problem of support vector machine becomes a constrained optimization problem as in Eq. 11.6 (The factor $\frac{1}{2}$ is used for mathematical convenience.)

$$\text{minimize: } Q(W) = \frac{1}{2}||W||^2 \tag{11.6}$$

$$\text{subject to: } y_i(WX_i + b) \geq 1 \qquad \forall (X_i, y_i) \in D \tag{11.7}$$

This type of constrained optimization problems are solved using **Lagrange multiplier method**. The above equations have a solution only when the data points are linear separable. It doesnt have a solution when the data points are not linearly separable. This version of SVM is called **Hard Margin SVM**. But in real world cases, data points are typically noisy, i.e the data points are not linearly separable. How do we solve this problem then? Solution is **Soft Margin SVM**.

- To deal with slightly non linear separable data, soft margin SVM allows mislabeled data points while still maximizing the margine size. To account the amount of misclassification it introduces slack variable (ζ_i) which measure the degree of misclassification.

The following is the optimization problem for soft margin SVM as given in Eq. 11.8 & Eq. 11.9 where C is a hyper parameter that determines the tradeoff between the margin size and the amount of error ($\zeta_i \geq 0$) in training

$$\text{minimize: } Q(W, b, \zeta_i) = \frac{1}{2}||W||^2 + C\sum_i \zeta_i \tag{11.8}$$

$$\text{subject to: } y_i(WX_i + b) \geq 1 - \zeta_i \qquad \forall (X_i, y_i) \in D \tag{11.9}$$

We can also interpret the SVM problem based on **hinge loss** (ζ_i). The hinge loss can be defined as Eq. 11.10 and is shown in Fig. 11.4.

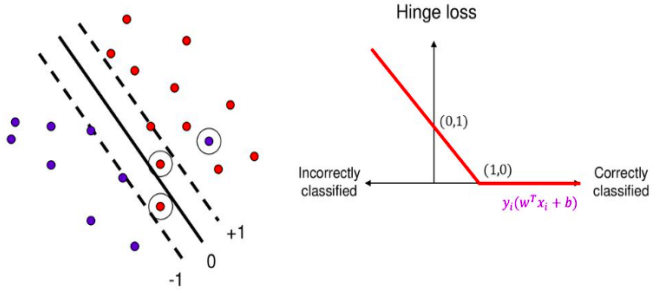


Figure 11.4: Hinge Loss

$$L = \zeta_i = max(0, 1 - y_i(W^T X_i + b)) \tag{11.10}$$

- For datapoints on correct side of the margin, the hinge loss is zero. Hinge loss is proportional to the *distance from the margin for datapoints on wrong side.*

We then wish to minimize hinge loss and the SVM training problem can be summarized as Eq. 11.11,

$$\hat{W}, \hat{b} = \arg\min_{W,b} \frac{1}{N}\sum_{i=1}^{N} \zeta_i + \lambda||W||_2^2 \quad \text{such that } y_i(W^T X_i + b) \leq \zeta_i \tag{11.11}$$

The parameter λ determines the tradeoff between increasing the margin-size and error ζ and it is similar to C hyperparameter in Lagrange multiplier method form.

In standard soft margin SVM, we have only one parameter C so it also known as C-**SVM**, there is a variant of this SVM called ν-**SVM** where it puts an upper bound on the error. Here $0 \leq \nu \leq 1$ is a hyperparameter that represents the fraction of error.

So far we have discussed SVM for binary classification. How do we extend the SVM to multi class classification problems? **One Versus Rest** method !!. This method uses many binary classifers like one versus the rest pairs.

11.2 Kernel Support Vector Machines

One last thing to discuss about SVM classifier is that how does it work on more complex non-linear separable data points. If the training data is not linearly separable, there is no hyperplane that can separate the classes. **Kernel functions !!** comes to the rescue.

Linear separability is achieved by projecting the data into higher dimensions where the data is lineraly separable as shown in Fig. 11.5. Kernel function uses different kernels like **polynomial kernal** and **radial basis function kernels** to project the data into higher dimensions.

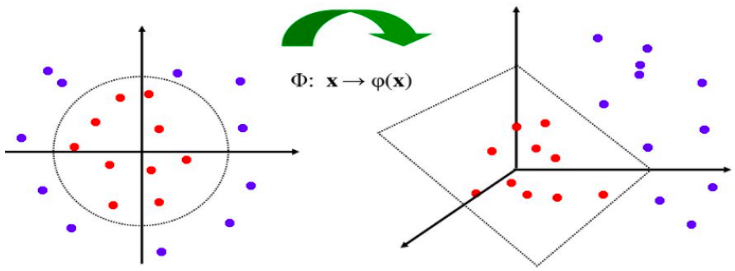


Figure 11.5: Projecting into Higher Dimensions

There are two steps in **kernal version** of SVM.

1. The input vectors are transformed into high-dimensional feature vectors where the data can be linearly separated.
2. SVMs are used to find the hyperplane of maximal margin in the new feature space. The separating hyperplane becomes a linear function in the transformed feature space but a nonlinear function in the original input space.

11.3 SVM-Regression

Support vector machine can be used for regression problems **SVR**. Like in classification prolblems, SVR uses the idea of maximum margin. Here we try to fit a hyperplane to the data. We want most of the data points within the margin (ϵ). The middle hyperplane is the best fit hyperplane for regression and the predicted value for input X_i is computed using the equation $F(X_i) = W.X_i + b$. This concept is shown in Fig. 11.6.

The **hard margin SVR** contraints for every input vector $X_i \in D$ are given below,

$$y_i - (WX_i + b) \geq \epsilon \tag{11.12}$$

$$(WX_i + b) - y_i \geq \epsilon \tag{11.13}$$

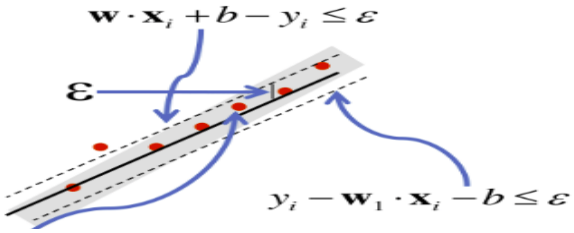


Figure 11.6: Support Vector Regression

The margin is

$$margin = \frac{1}{||W||} \tag{11.14}$$

By minimizing $||W||^2$ to maximize the margin, the training problem in SVR becomes a constrained optimization problem as follows.

$$\text{minimize}: \ L(W) = \frac{1}{2}||W||^2 \tag{11.15}$$

$$\text{subject to:} \ y_i - (WX_i + b) \geq \epsilon \tag{11.16}$$

$$(WX_i + b) - y_i \geq \epsilon \tag{11.17}$$

As most of the real world datasets are noisy, we may need to allow some data points outside the margin, to do this **soft margin SVR** version introduces slack variables ζ, ζ^*. Then, this optimization problem can be revised with slack variables and the concept is shown in Fig. 11.7.

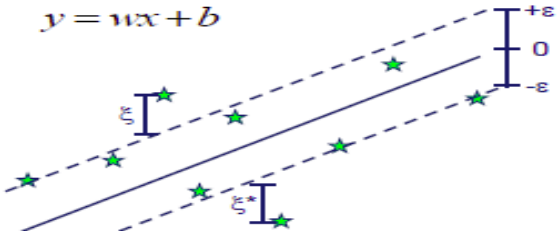


Figure 11.7: Softmargin SVR

$$\text{minimize}: \ L(W, \zeta) = \frac{1}{2}||W||^2 + C\sum_{i=1}^{N}(\zeta_i^2, \zeta_i^{*2}) \tag{11.18}$$

$$\text{subject to:} \ y_i - (WX_i + b) \geq \epsilon + \zeta_i \qquad \forall(X_i, y_i) \in D \tag{11.19}$$

$$(WX_i + b) - y_i \geq \epsilon + \zeta_i^* \qquad \forall(X_i, y_i) \in D \tag{11.20}$$

$$\zeta_i, \zeta_i^* \geq 0, C > 0$$

- The constant $C > 0$ is the trade-off parameter between the margin size and the amount of errors, slack variables ζ and ζ^* imposes the penalty to the excess deviations which are larger than ϵ.

To solve this contraint optimization problem Lagrange multiplier method is used. Like classifier model, Kernel trick is used for dealing with non-linear data points.

11.4 Cases

- Once the SVM is trained with training data, the complexity of the classifier is characterized by the number of support vectors due to this dimensionality of data is not an issue in SVM unlike in other classifiers.
- There is no easy way to get important features from SVM.
- Model is interpretable for linear svm but for kernel SVM.
- It is less sensitive to outlier since we use support vectors.
- Bias variance trade off : As C increases it causes overfit since we multiply C with loss and C decreases causes underfit.
- Linear SVM is scalable to large dataset but the kernel SVM.

12

Decision Trees

A decision tree is a model composed of a collection of *questions* organized hierarchically in the shape of a tree. The questions are usually called a **condition**, a **split**, or a **test**. Each non-leaf node contains a condition, and each leaf node contains a prediction as shown in Fig. 12.1.

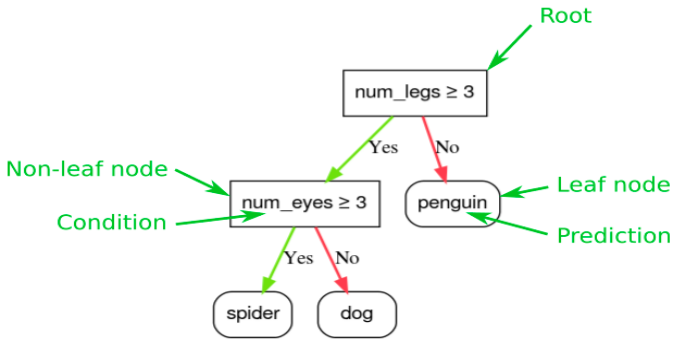


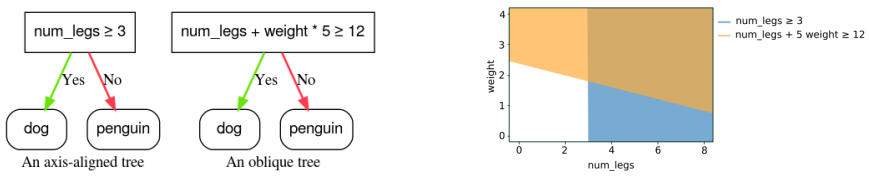
Figure 12.1: Decision Tree

- Once decision tree is build from data, the inference is computed by following a path from root to the leaf which contains a prediction value for either classification or regression.
- The path followed from root to leaf node is the **inference path** which is easily interprettable.
- Tress can be build with only two branch per node (**binary decision tree**) or more than two branch per node (**non-binary decision**). Conditions with too branches are more likely to overfit. For this reason, decision forests generally use binary decision trees.
- A non-binary condition can be emulated with multiple binary conditions, therefore, binary trees are not inherently less powerful than non-binary trees.

- A decision tree with depth one is called a **decision stump**.

Two types of **conditions** used to build decision trees are show in Fig. 12.2.

1. **axis-aligned condition** : This involves only a single feature and easy to implement.
2. **oblique condition** : This involves multiple features, usually the performance is better than axis aligned version but more complex to impement and costlier to train in terms of execution time.



(a) Programming - nested if else (b) Linear Algebra - hyperplanes

Figure 12.2: Condition Types - Interprettation

The conditos used in decsion tree can be put into various categories:

- **Equality condition** : eg : species = 'cat'
- **Threshold condition** : eg: price ≥ 30, weight ≤ 75
- **In-Set condition** : eg: species $\in$ "cat', 'dog', 'bird'
- **Oblique condition** : eg: price ≥ 30 & weight ≤ 25
- **Feature is missing** : eg: if *price_missing*

12.1 Building a Decision Tree

The optimal training of a decision tree is an NP-hard problem. Therefore, training is generally done using heuristics—an easy-to-create learning algorithm that gives a non-optimal, but close to optimal, decision tree. Most algorithms used to train decision trees work with a greedy divide and conquer strategy. The algorithm starts by creating a single node (the root) and recursively and greedily grows the decision tree. Other methods exist to grow decision trees - popular alternative is to optimize nodes globally.

- At each node, all the possible conditions are evaluated and scored. The algorithm selects the *best* condition, that is, the condition with the highest score.

- The algorithm then repeats recursively and independently on both children nodes. When no satisfying conditions are found, the node becomes a leaf node.

Depending on the number and type of input features, the number of possible conditions for a given node can be huge, generally infinite. For example, given a threshold condition $feature \geq threshold$, the combination of all the possible threshold values (**split points**) for $threshold \in \mathbb{R}$ is infinite. If the feature is discrete the we have **split-subsets** to evaluate for in-set condition. The number of such subsets to evaluate is exponential in nature.

- The routine responsible for finding the best condition is called the **splitter**. Because it needs to test a lot of possible conditions, splitters are the bottleneck when training a decision tree. There are different splitter algorithms each with varying support for:
  - Feature type - Categorical, Numerical etc.
  - Condition type - Threshold, Equality etc.
  - Task - binary classification, multi-class classification, regression etc.
- Splitters do optimizes a score while splitting a dataset and the score to optimize depends on the problem at hand,
  - classification problems : **Information Gain**, **Gain Ratio**, **Gini Index**. The labels are set by majority vote on labels in the leaf node.
  - regression problems : **Mean Squared Error**. Labels are set by taking the mean value of labels in the leaf node.

Our objective while training a decision tree is to find a condition value (a threshold or a set for categorical features) on a feature that would split the data set into 2 partitions so that the resulting partitions would improve the task of separation of the labels for classification problems and predicting a value for continuous output for regression problem.

- The two partitions we get are:
  - True set T : This set contains all elements that satisfy the given condition
  - False set F : This set contains all elements that doesnt satisfy the given condition
- By partitioning the dataset based on a condition, we aim to reduce the impurity - the impurity or radomness can be measured using entropy.

- By partitioning, we are expecting the resultant partitions are easier to separate than the original partition - meaning the sum of entropies of resulting partition is less than the original partition.
- The entropy difference between D and the T, F set is called the **information gain**, so we want to maximize the information gain.

Let X denote an instance of the dataset and n_{pos} be the number of positive examples in X, so the probability of of a record in X belongs to positive class is $p = \frac{n_{pos}}{|X|}$ and $1 - p$ is the probability of negative sample.

Entropy $H(D)$ of dataset D and information gain $IG(D)$ for the split on D into two subsets T & F are given below.

$$H(X) = p * \log\left(\frac{1}{p}\right) + (1-p) * \log\left(\frac{1}{1-p}\right) \tag{12.1}$$

$$IG(D) = H(D) - \left(\frac{|T|}{|D|} * H(T) + \frac{|F|}{|D|} * H(F)\right) \tag{12.2}$$

How do we search for the threshold values for a continuous feature from an infinite possible values for thresholds? Though we have infinite number of thresholds possible, in practice we have only finite number of data points and hence finite number of intervals to test for.

- We can sort the feature values in ascedning order and then take the mid point of every consequtive values and evaluate the conditions on those mid points.
- We apply this splitter algorithm to parent node and its children. Each child node get around $\frac{1}{2}$ of the samples in parent node. So with N samples and d features with sorting algorithm of complexity $O(N \log N)$, this takes $O(dN \log^2 N)$. The space complexity is proportional to the depth of the tree.
- In this algorithm the feature values doesn't matter, only the order of feature values matter, so scaling has no impact on the algorithm.

Decision trees use attribute selection measures to score each feature and use that score to decide the best feature to be used for splitting the data set. We have seen one metric called information gain for classification models, other mesure for classification model is the **Gini Impurity**.

The Gini impurity of a parent node denoted as $GI(X)$ is computed using the Gini impurities of both children $GI(T)$ & $GI(F)$. This is given in Eq. 12.3 and depicted in Fig. 12.3, where k represents a class label (0-negative & 1-positive), p_k denote the probability of a class label in the dataset T or F.

$$GI(T) \textbf{ or } G(F) = 1 - \sum_{k=0}^{1} p_k^2 \tag{12.3}$$

$$GI(X) = \frac{|T|}{|X|} * GI(T) + \frac{|F|}{|X|} * GI(F) \tag{12.4}$$

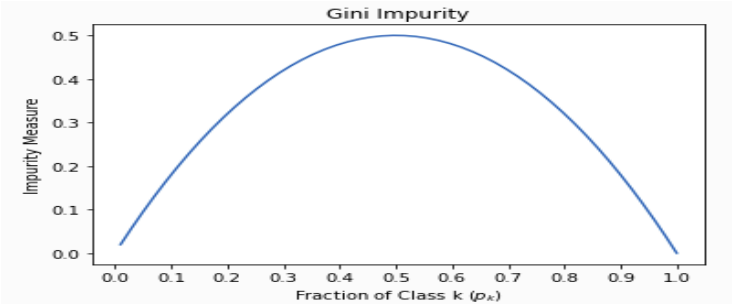


Figure 12.3: Gini Impurity

- The gini impurity has minimum value 0 when probability of a class is either 0 or 1 and has the maximum value 0.5 when the class labels are equiprobable (uniform distribution).
- Like information gain, the gini gain can be computed for a split by subtracing the weighted sum of gini impurities of children from the parent node. The feature with highest Gini gain or minimum gini impurity is selected as the feature for splitting.
- Gini impurity is cheap to compute (compute the square value) when compared with information gain that uses log. So gini index is popular over information gain.

Is there any problem with Information Gain or Gini impurity? Are there any edge cases that could cause problems with these metrics? Yes.

- Suppose we have one categorical feature in the data feed with unique values, we will be always able to partition this type of feature into two non overlapping subsets such that one set contains all values corresponding to positive labels and the other set contains all values related to negative samples.
- When we evaluate information gain or gini impurity, these partitions have 0 impurity and the metrics favour this partition. But the unique features may not help the model to generalize rather it will overfit. For eg: The unique feature could be a customer id for a transaction that is not useful for generalization.

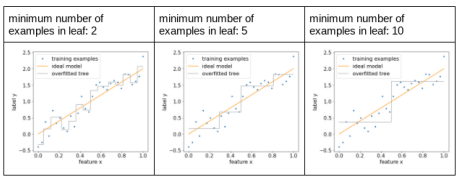
So far we have seen the decision tree can be used for classification. How do we use a decision tree for regression?

- A decsion tree can be converted to regression tree by using mean squared error as a score for attribute selection or any other metric that works for a regressor.
- Predicted values are the average value of all labels that fall in a leaf node.

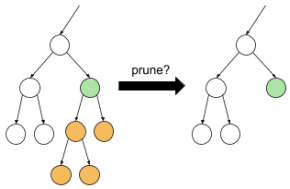
12.2 Overfitting, Interprettation, Feature Importance

Noise often causes problem in training a model. If the dataset contains noise tree will overfit to the data and show poor test results. To limit overfitting a decision tree, apply one or more of the following regularization criteria while training the decision tree. These methods are shown in Fig. 12.4.

- **Minimum number of samples in leaf** : A node with less than a certain number of examples will not be considered for splitting further.
- **Maximum depth** : Prevent decision trees from growing past a maximum depth. Generally higher depth causes overfitting and low depth leads to underfit.
- **Tree Pruning** : Selectively removing (pruning) certain branches of the tree. If removing the branch improve the model performance on a validation set, then that branch is converted from non-leaf to a leaf.



(a) Minimum leaf samples



(b) Pruning

Figure 12.4: Handling Overfitting in Trees

- Decision tree is easily interprettable. Any path from root to leaf node gives a reason for prediction.
- Variable importance (feature importance) of a feature indicate how important is the feature in prediction. Decision trees have specific variable importances. Below criterion indicate a feature is important when the,

  - *number of nodes* are high with a given variable - this indicates how much a decision tree is looking at this specific feature with a condition, which might indicate variable importance. However, the same feature appearing in multiple conditions might also indicate that a model is trying but failing to generalize the pattern of a feature. For example, this can happen when a feature is just an example identifier with no information to generalize.
  - *average depth* of the first occurrence of a feature across all the tree paths is low
  - *sum of split score* is high for a feature

On the other hand, a high value for a high permutation variable importance (a generic feauture importance test) indicates that removing a feature hurts the model, which is an indication of variable importance. However, if the model is robust, removing any one feature might not hurt the model.

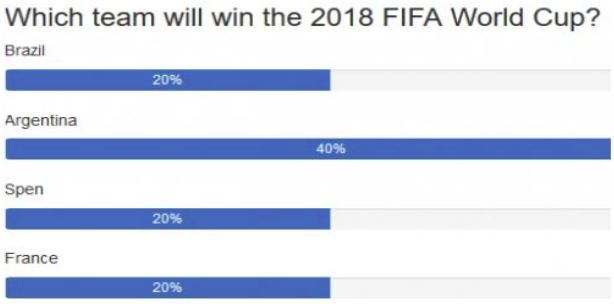
Because different variable importances inform about different aspects of the models, looking at several variable importances at the same time is informative. For example, if a feature is important according to all the variable importances, this feature is likely important. As another example, if a feature has a high "number of nodes" variable importance and a small "permutation" variable importance, then this feature might be hard to generalize and can hurt the model quality.

- Decision tree has natuaral extension to multiclass problems.
- Imbalanced data badly affect the entropy, MSE, MSD calculations.
- Large dimesnion dramatically increases time complexity as we need to do the evaluate the score of each attribute.

13

Ensemble Models

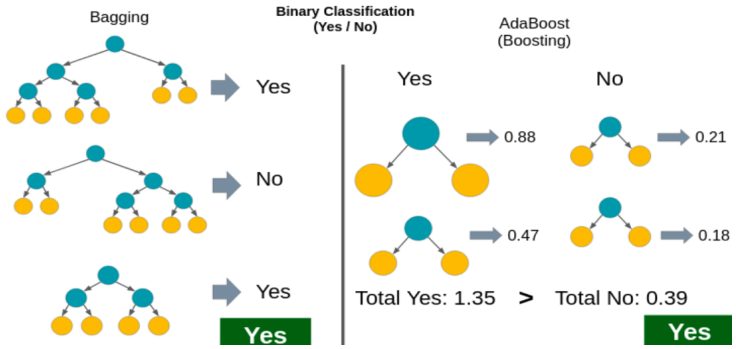
We regularly come across many audience polls where we need to vote for one option among a set of other options. Most of the time contestents will go with what the majority said and they succeed in it. The fundamental idea of ensemble models are based on this.



The underlying principle of ensemble learning is a recognition that in real-world situations, every model has limitations and will make errors. Given that each model will have these 'limitations' the aim of ensemble learning is to manage their strengths and weaknesses, leading to the best possible decision being taken overall. So ensemble models are based on a group of individual models and hence also known as **committee-based models** or **composite model**. An ensemble model can be thought of *wisdom of the crowd*, in certain situations, collective opinion provides very good judgment.

- Most ensemble methods use a single base learning algorithm to produce **homogeneous base learners** but there are also some methods which use multiple learning algorithms to produce **heterogeneous learners**.
- To get a good ensemble, it is generally believed that the base learners should be as accurate as possible, and as diverse (**little correlation** among base learners) as possible.

- By combining base learners, ensemble model aims to achieve a strong generalization ability and robustness against noise. The benefits of combining models can be summarized as follows:
  - In many problems the unknown hypothesis could not be represented by any single hypothesis. Combining task would help us to acheive more approximation to the true hypothesis and hence reduces the bias problem.
  - We will be more confident in the prediction when we have support from more number of learners. So this will help the model to predict well on unseen data. So combination of learners reduces the variance.
  - Single learners does a local search on the hypothesis space, but combining learners have multiple starting points to search the space and hence higher the chances for finding global optimum solutions.
- According to how the base learners are generated, there are two types of ensemble models.
  - **Parallel ensemble methods** : Based on exploiting the independence of models. So model training can be distibuted and the training speed can be easily accelerated using multi-core computing processors. eg: **Bagging** models
  - **Sequential ensemble methods** : Based on exploiting the dependency between the base learners. Overall performance of the ensemble model can be improved by considring the error / gap in the learning from the previous step. **Boosting** models



Most of the ensemble models are based on trees because, it is a non-linear model, interprettable, scalable and it is relatively easy to combine tree based learners. Group of many decision trees forms the **Decision forest**.

The prediction of a decision forest is the aggregation of the predictions of all trees. The implementation of this aggregation depends on the algorithm - For example, in random forest algorithm, each tree votes for a single class, and the prediction is the most represented class. In gradient boosted Tree (GBT) algorithm, each tree outputs a floating point value, and the prediction is the sum of those values followed by an activation function.

13.1 Random Forest - A bagging algorithm

Random Forest is an ensemble of decision trees in which each decision tree is trained with a specific random noise. Random Forest is based on **Bagging** which is the short form of *Bootstrap AGGregatING* i.e **bootstrap** followed by **aggregation**. Here each decision tree is trained on a different subset of examples from training dataset. By doing so we get the base learners as independent as possible.

- In bagging, each decision tree is almost always trained on the total number of examples in the original training set. Training each decision tree on more examples or fewer examples tends to degrade the quality of the random forest.
- While not present in the original random forest paper, the sampling of examples is sometimes done "without replacement"; that is, a training example cannot be present more than once in a decision tree training set.

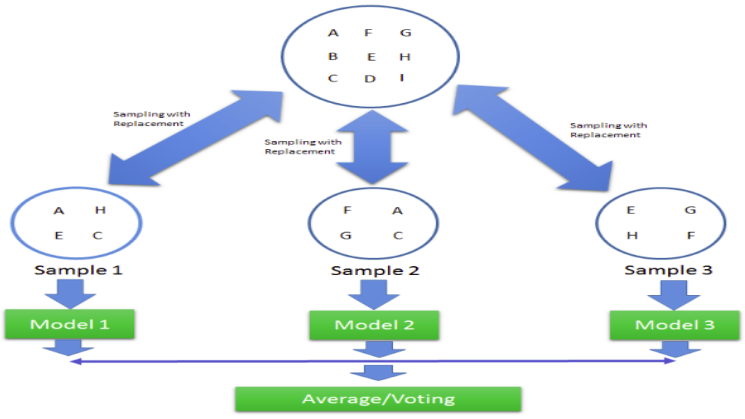


Figure 13.1: Random Forest Building

- The generalization error of the base learner can be estimated by cross validation using the datapoints which are not selected in bootstraping (**out-of-bag examples**).

- Change in training dataset (bootstrap sampling) impact only few subset of models. So bagging reduces the variance without impacting the bias much.
- In bagging we take bunch of low bias high varince base learners and combine them to get low bias reduced variance model.
- It is worth mentioning that though the error might not drop to zero, the performance of Bagging converges as the ensemble size, i.e., the number of base learners, grows large.

Instead of looking for the best condition over all available features, only a random subset of features are tested at each node. This is known as the **Column sampling** or **Attribute sampling**. In contrast to the classic bagging based implementation of Random Forest, many implementation uses attribute sampling.

Fig. 13.2 shows this, where blue cells represent selected attrubutes, white cells represent the ignored one and the red highlighted one represent the best attribute selected for node splitting.

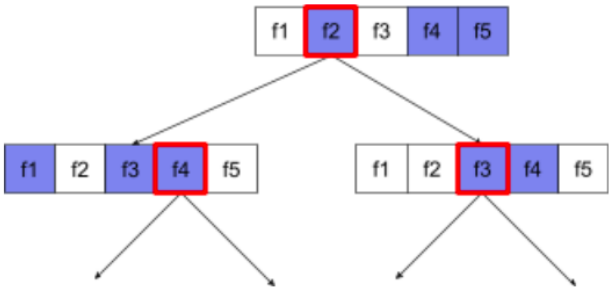


Figure 13.2: Attribute Sampling

- Ratio of the attribute sampling is one of the important feature in random forest. Many random forest implementations test, by default, $\frac{1}{3}$ of the number features d for regression and $\sqrt{d}$ for classification.
- The decision trees in a random forest are trained without pruning. The lack of pruning significantly increases the variance and significantly reduces the bias of the individual decision tree learning. In other words, the individual decision trees overfit, but the random forest is not.
- Due to the overfit of individual trees, a very high training score in a random forest is normal and does not indicate that the random forest is overfitted.

- The two sources of randomness (bagging and attribute sampling) ensure the relative independence between the decision trees. This independence corrects the overfitting of the individual decision trees. Consequently, the ensemble is not overfitted.

- Pure random forests train without maximum depth or minimum number of observations per leaf limits. In practice, limiting the maximum depth and minimum number of observations per leaf is beneficial.

Why would random noise improve the quality of a random forest? To illustrate the benefits of random noise, Fig. 13.3.shows the predictions of a classical (pruned) decision tree and a random forest trained on a few examples of simple two-dimensional problem with an ellipse pattern. Ellipse patterns are really hard for decision trees to classify based on the axis parallel lines separating boundaries.

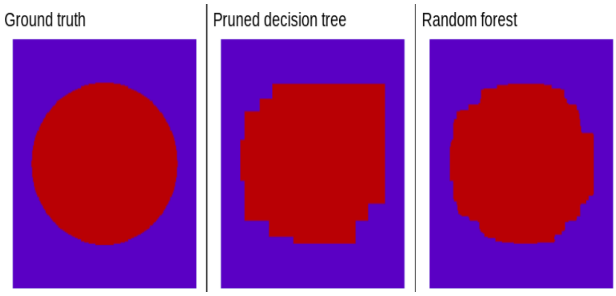


Figure 13.3: Ellipse Classification Example

- Because the decision trees of a random forest are not pruned, training a random forest does not require a validation dataset. In practice, and especially on small datasets, models should be trained on all the available data.

- When training a random forest, as more decision trees are added, the error almost always decreases; that is, the quality of the model almost always improves. At some point, the model just stops improving. In other words, adding more decision trees cannot cause the random forest to overfit.

13.1.1 Out of Bag examples & Interpretation

Random forests do not require a validation dataset. Most random forests implementation uses a technique called **out-of-bag-evaluation** to evaluate the quality of the model, i.e the samples that doesn't get into the bootstrap sample end up forming the validation set.

- To evaluate the model, for a sample we can use the trees that did not see this example during training step.
- Out of Bag evaluation is also effective to compute permutation variable importance for random forest models. Permutation variable importance (a model agnostic measure) measures the importance of a variable by measuring the drop of model quality when this variable is shuffled. Some of the permutation variable importances are mean decrease in accuracy, auc, precision recall-auc etc.

Random forests are more complex to interpret than decision trees. Random forests contain decision trees trained with random noise. Therefore, it is harder to make judgments on the decision tree structure. However, we can interpret random forest using **SHAP** (**SHapley Additive exPlanations**) is a model agnostic method to explain individual predictions or model-wise interpretation.

Some of the pros and cons and points to be noted on Random Forest are:

- Like decision trees, random forests support natively numerical and categorical features and often do not need feature pre-processing.
- Because the decision trees are independent, random forests can be trained in parallel. Consequently, you can train random forests quickly.
- Because decision trees are not pruned, they can be large. Models with more than 1M nodes are common. The size (and therefore inference speed) of the random forest can sometimes be an issue.
- The train time, test time complexities are $O(N \log N * R)$, $O(R *$ depth of tree) respectively and space complexity depends on the number of trees.

13.2 Boosting

Any weak learner is potentially able to be boosted to a strong learner. This is the basic idea of boosting algorithms.

- **weak model** : typically a decision tree model with high bias. A model that almost underfit.
- **strong model** : composed of multiple weak models and the combined performance of weak models is close to perfect performance i.e low bias and low variance. By combining models we reduce the bias at the same time keeping the variance low.

In gradient boosting, at each step, a new weak model is trained to predict the "error" of the current strong model. The error here referes to the difference between the target label and the model prediction and also known as the **psedo response** or **psedo residual**. The weak model (that is, the "error") is then added to the strong model with a negative sign to reduce the error of the strong model.

Gradient boosting is iterative. Each iteration invokes the following formula where F_i denote the strong model and f_i denote weak model at step i. The iterative procedure continues until the maximum number of iterations has reached or if the (strong) model begins to overfit as measured on a separate validation dataset.

$$F_{i+1} = F_i - f_i \tag{13.1}$$

The following code snippet shows the general naive boosting method using decison trees

Algorithm 1 General Boosting Procedure

Input

X: Feature set $\{(x_i)\}_{i=1}^{N}$

M: Number of models in ensemble

Y: Labels

Output:

$F_M(X)$: Strong model

1: Initialize forest as empty list & initial strong model predictions ($F_0(X)$) as zeros

forest = list()

$F_0(X)$ = np.zeros(Y)

2: **for** $m = 1$ to M **do**
3: Compute error / residual for model m $r_m = F_{m-1} - Y$
4: Create a new weak model f_m = Model(...)
5: Fit weak model to residual f_m.fit(Y, r_m)
6: forest.append(f_m)
7: Update the strong prediction $F_m(X) = F_{m-1}(X) - f_m(X)$
8: **end for**
9: return F_M as the strong model

The above general boosting algorithm lacks the usage of **Shrinkage** ($\nu \in [0, 1]$) which controls how fast a model learns. This is analogus to learning rate in neural network. Shrinkage is used in the update stage of boosting as $F_m(X) = F_{m-1}(X) - \nu f_m(X)$. Shrinkage value closer to 0.0 reduces overfitting more than a shrinkage value closer to 1.0.

13.2.1 Gradient Boosting

In Gradient boosting method, we define a loss function to compute the gradient or error and use that gradient to update the model.We train the weak model to predict the gradient of the loss according to the strong model output.

Formally, given a loss function $L(y, F_m(X_i))$ where y_i is a label for i^{th} example and $F_m(X_i)$ is the prediction of strong model on i^{th} exmple, the pseudo response (z_i) used to train the weak model at step m is:

$$z_i = \frac{\partial L(y, F_m(X_i))}{\partial F_m(X_i)} \tag{13.2}$$

- For regression problem, when we use mean squared error as loss function, the gradient is signed error. This equivallence of signed error doesn't hold when we use cross entropy loss for classification problems.
- We use ν shrinkage factor which makes the constant multiplier values in the gradient negligible.

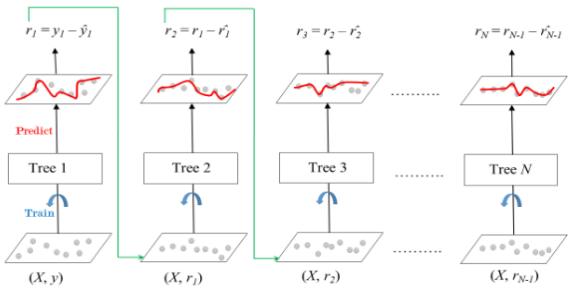


Figure 13.4: Gradient Boosting for Regression

Since we use shallow tress in boosting, each leaf node holds more samples when compared with random forest. So the minimum number of nodes parameter for leaf nodes in boosting has no significant impact. Unlike random forests, gradient boosted trees can overfit. Therefore, as for neural networks, you can apply regularization and early stopping using a validation dataset. Common regularization parameters for gradient boosted trees include:

- The maximum depth of the tree
- The shrinkage rate
- The ratio of attributes tested at each node
- L_1 and L_2 coefficient on the loss

Some pros and cons of gradient boosting are as follows:

- Like decision trees, they natively support numerical and categorical features and often do not need feature pre-processing.
- Gradient boosted trees have default hyperparameters that often give great results. Nevertheless, tuning those hyperparameters can significantly improve the model.
- Gradient boosted tree models are generally small (in number of nodes and in memory) and fast to run (often just one or a few μs / examples).
- The decision trees must be trained sequentially, which can slow training considerably. However, the training slowdown is somewhat offset by the decision trees being smaller.

13.3 Stacking - Combining by learning

Stacking is a general procedure where a learner is trained to combine the individual learners. In stacking, the individual learners are called the **first-level learners**, while the combiner is called the **second-level learner**, or **meta-learner**. Here we expect the individual first level learners to be strong learners and as diverse as possible (heterogeneous ensemble). The basic idea of stacking is shown in Fig. 13.5.

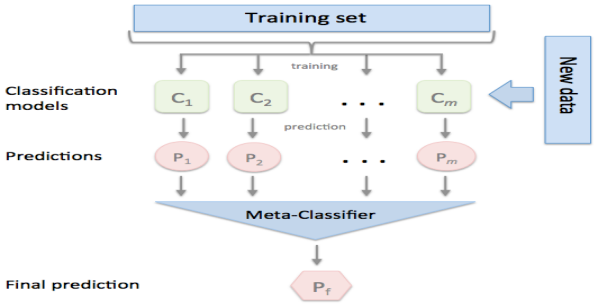


Figure 13.5: Stacking Model

- For classification problems, usually class probabilities (instead of crisp class labels) are used as featues to train the meta learner. This makes it possible to take into account not only the predictions but also the confidences of the individual classifiers.
- For regression problems, the regressed values can be directly taken as the features.

14

Cluster Analysis

Imagine you have 10M customers, and you want to develop customized or focused marketing campaigns. It is very unlikely that you will develop 10M marketing campaigns, so what do we do here? We can group 10M customers into say 25 groups (or **clusters**) based on customers behaviour like(income of customers, amount spend, frequency of purchase etc) and then run 25 marketing campaigns. This is *clustering*.

- Clustering is **learning by observation**, i.e we do not have any labels to group data points - *unsupervised learning*.
- We use a similarity metric to group the data points such that objects within a cluster have high similarity (intracluster similarity), but are dissimilar to objects in other clusters (intercluster dissimilarities).
- In clustering we assign a **cluster-id** to each data point, that makes the handling of data points easier.
- Data professionals often use clustering in the Exploratory Data Analysis phase to discover new information and patterns in the data.

Cluster analysis or **clustering** is used in various industries for wide variety of applications and some of them are:

- **e-commerce** : In e-commerce sites, customers will give reviews about a product. Manually lablelling all these reviews is time consuming and very costly when we have millions of reviews. Clustering is far cheap than complete manual labelling method.
- **NLP** : We can group news feed into several topics called Topic modelling.
- **Medical**: Clustering is used in medical field to group pixels of images like MRI scan to detect tumors, Alzheimer's disease etc. This is an application of image segmentation.

There are many clustering techniques used in the industry and three popular methods are **Centroid Based Method**, **Hierachical Method** & **Density Based Method**. These popular clustering algorithms are discusssed in the following sections. In this chapter we use some notation. Let D be a data set with N objects with d features and let K be the number of clusters we need where $(K << N)$. Objects are denoted as x and clusters are denoted as C_is where C_i is the i^{th} cluster.

- When choosing a clustering algorithm, we should consider whether the algorithm scales to our dataset.
- Many clustering algorithms work by computing the similarity between all pairs of examples. This means their runtime increases as the square of the number of examples N, so the complexity is $O(N^2)$. These type of algorithms are not practical when number of data points are in the range of millions.
- Handling missing values is more challenging in clusterng. We cannot easily put "Unkown" for missing values as the similarity between unknown and a number is not well defined.

Four stages involved in the clustering process are shown in Fig. 14.1.

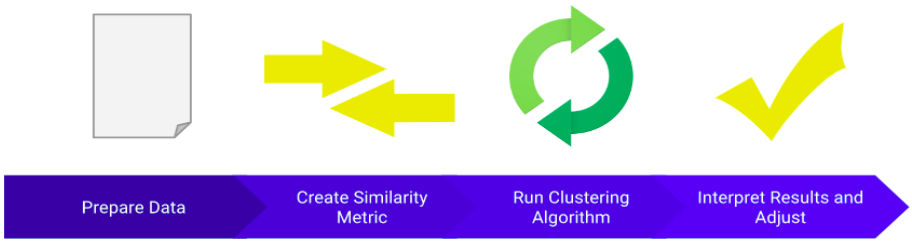


Figure 14.1: Clustering Steps

14.1 Preparing the Data & Similarity Measure

We need to prepare the data before we do clustering. As we use similarity metric for a pair of data points, we need to do the proper scaling and transformation. We must ensure that we will be able to accuarately measure the similarity between pair of points.

- Combining all the features of examples into a numeric value requires that the all features have to be on same scale.
- Standard scaling can be used if the data follows normal distribution or we lack enough data points and log trasformation if the data follows log normal distribution

- If the data doesn't follow specific distribution like normal or log normal, we can use more genric method like quantiling. To apply the quantile method, we need sufficiently large number of data points. As a rule of thumb we need the number of data points to be 10 times the number of quantiles.
  - In quantiling we decide the number of intervals where each number of intervals have roguhly same number of data points
  - We assign an index to each data points based on the interval it belongs
  - We scale the index to fit in the range $[0, 1]$, for all featues and then compute the similarith between pair of points.

Creating a similarity metric requires us to carefully understand the data and how to derive similarity from the features. To calculate the similarity between two examples, you need to combine all the feature data for those two examples into a single numeric value.

Suppose we want to compare two shoes by its size, the we can compare the shoes directly by checking the difference in size. In this case the similarity measurement is easy and we can manually set the similarity measure for size feature this method is called **manual similarity measure**. But if you want to compare two shoeds for similarity based on both size and colour, what do we do? Colour is a categorical variable and its hard to compare with a numerical type. In this case we can switch to a similarity measure called **supervised similarity measure**. We can always use supervised similarity measure like embedding when we have trouble in using manual similarity measure.

- With manual similarity measures, we can give different weightage to different features. We can compute the feature wise differences and compute the RMSE values. When we consider the house price dataset, color should be given low weightage than the number of bedrooms.
- If you create a similarity measure that doesn't truly reflect the similarity between examples, your derived clusters will not be meaningful. This is often the case with categorical data and brings us to a supervised measure.
- Gaining insights from manual measure is far easy than from a suprervised similarity measures.

Supervised similarity measures are often created using embeddings. Instead of comparing manually-combined feature data, you can reduce the feature data to representations called embeddings, and then compare the embeddings. These embedding are usually created by autoencoders & DNN prediction models

14.2 Centroid Based Methods

Centroid based (also known as partitioning based) methods groups data points into several **non-overlapping clusters**. Two most well-known and commonly used are **k-means** and **k-medoids**. In this type of clustering, the dataset is divided into a set of K groups. The cluster center (**centroid**) is created in such a way that the distance between the data points of one cluster is minimum as compared to another cluster centroid. Cluster C_i is represented by the centroid (c_i), which can be either mean of all data points in C_i or **medoid** (a representitive point) of C_i.

How do we solve this centroid based clustering problem?

- Optimizing the within-cluster variation/similarity is computationally challenging. To see why, lets look at the below scenario
  - In the worst case, we would have to enumerate all possible partitionings of N examples into K groups. This enumeration is an NP-Hard problem.
- So we solve the centroid based clustering problem using an **approximation algorithms** called **Lloyds algorithm** which forms the basis of k-means algorithm.

14.2.1 k-means: centroid-based clustering

k-means groups points into clusters by minimizing the distances between points and their cluster's centroid. The centroid of a cluster is the mean of all the points in the cluster. Before running k-means, you must choose the number of clusters K. Initially, we start with a guess for K and will tune it. So K is the hyperparameter.

- k-means finds roughly circular clusters. Conceptually, this means k-means effectively treats data as composed of a number of roughly circular distributions, and tries to find clusters corresponding to these distributions.

To cluster the data points k-means follows the below Algorithm 2:

k-means scales as $O(TKN)$, linear - so it scale for large dataset. Where T is the number of iterations which is fixed. The four major steps of k-means are shown in the below Fig 14.2.

Now lets look into the objective function of k-means algorithm and mathematical proof why the algorithm works. Let F_{ij} is a flag variable that takes the value 1 if object i belongs to cluster j and 0 otherwise. Let c_j be the centroid of cluster C_j. Here we want to minimize the following function.

Algorithm 2 K-means clustering

Input:

K: Number of clusters

D: Data set containing N objects

Output:

Set of K clusters

1: **Initial step** : Arbitrarily choose K objects from D as the initial cluster centroids.
2: **repeat**
3: **Assignment stage** : (re) assign each point to the closest centroid to get k clusters.
4: **Recompute centroid** :recomputes all the centroids by taking the mean of all points in the cluster and update all the centroids.
5: **until** datapoints stops changing the clusters (convergence of algorithm)

$$\min_{F,c} \sum_{i=1}^{N} \sum_{j=1}^{K} F_{ij} ||c_j - x_i||^2$$

subject to:

$$F_{ij} \in \{0, 1\} \forall i, j$$

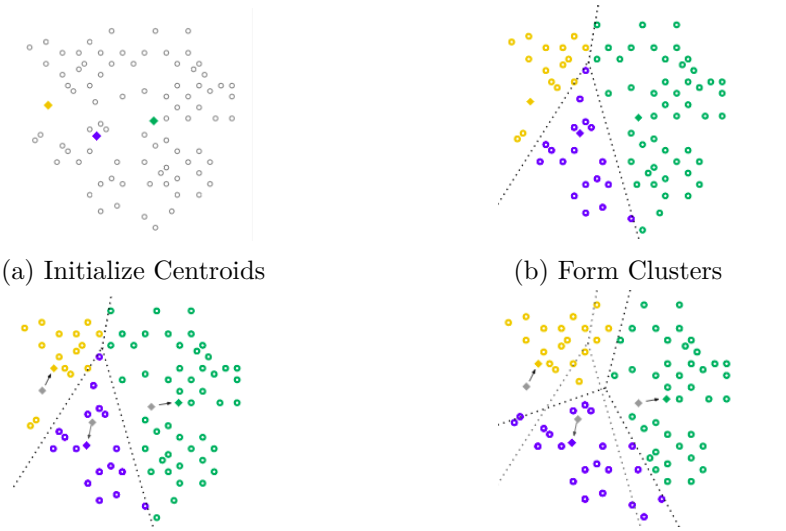
and

$$\sum_{j=1}^{K} F_{ij} = 1 \forall i$$

The above equation only considers the intracluster distance and it doesnt consider the intercluster distance (not the best possible solution but an approximation to best solution). To minimize the expression with respect to the cluster centroids c_j, take the derivative with respect to c_j and equate it to 0.

$$f(c) = \sum_{i=1}^{N} \sum_{j=1}^{K} F_{ij} ||c_j - x_i||^2$$

$$\frac{\partial f}{\partial c_j} = 2 \sum_{i=1}^{N} F_{ij}(c_j - x_i) = 0$$



(a) Initialize Centroids (b) Form Clusters

(c) Recompute Centroid (d) Convergence of Centroid

Figure 14.2: k-means Algorithm 4 Steps

$$\implies \sum_{i=1}^{N} F_{ij} c_j = \sum_{i=1}^{N} F_{ij} x_i$$

$$c_j \sum_{i=1}^{N} F_{ij} = \sum_{i=1}^{N} F_{ij} x_i$$

$$c_j = \frac{\sum_{i=1}^{N} F_{ij} x_i}{\sum_{i=1}^{N} F_{ij}}$$

The numerator is the sum of all example-centroid distances in the cluster. The denominator is the number of examples in the cluster. Thus, the cluster centroid (c_j) is the average of example-centroid distances in the cluster. Hence proved.

- The k-means algorithm is sensitive to outliers. This effect is particularly exacerbated due to the use of the **squared-error function**.
- K-means algorithm has a problem called **Initial sensitivity**. Final clusters and centroids depends on initial centroids. An elegant way of solving this problem is k-means ++, where we set the centroid using a probabilistic approach.
- k-means assumes the clusters are of spherical shape and have roughly equal size, if this condition is not met the results may be poor.

- k-means assumes data have same densities, it cannot handle varying densities.

"How can we modify the k-means algorithm to diminish the sensitivity to outliers or use it where cluster mean cannot be computed" ? Instead of taking the mean value of the objects in a cluster as *a reference point*, we can pick actual objects to represent the clusters, this algorithm is a variant of k-means called **k-medoids**.

- Here for recomputing the centroid we probabilistically select a new reference and check swapping this point with previous centroid improves the similarity score or not
  - if yes we swap, otherwise we will choose another reference point.
  - **Partitioning Around Medoid** is a popular method that falls into this category.
- Another variant of k-means algorithm to address the sensitivity to outlier is k-means++, where we pick the centroid using a probabilistic approach.

14.3 Hierarchical Clustering

In this type of clustering, we develop hierarchy of clusters by a sequence of merges or splits in the dataset. Clusters so formed can be visulaized as a tree called **dendrogram**. We have centroid based clustering methods, then why do we need hierarchical clustering?

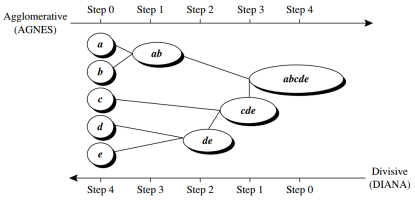
- In contrast to centroid based clustering, hierarchical clustering can form non-spherical clusters and cluster of different sizes. Also we don't need to specify the number of clusters in advance, we can decide on the number of clusters from the dendrogram.
- When it comes to analyzing data from social networks, hierarchical clustering is by far the most common and popular method of clustering. The nodes (branches) in the tree are compared to each other depending on the degree of similarity that exists between them.

The hierarchical clustering technique has two approaches. Fig. 14.3 shows these approaches applied on a data set with objects {**a, b, c, d, e**}.

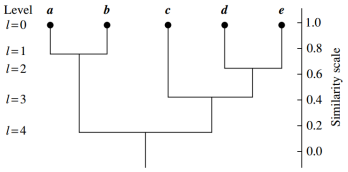
1. **Agglomerative** : bottom-up (merging)
   - Agglomerative method typically starts by letting each object form its own cluster and iteratively merges clusters into larger and larger clusters, until all the objects are in a single cluster or certain termination conditions are satisfied.

2. **Divisive** : top down (splitting)

    - This method starts by placing all objects in one cluster then strats dividing it into several smaller subclusters, and recursively partitions those clusters into smaller ones. The partitioning process continues until each cluster at the lowest level is coherent enough.



(a) Agglomerative & Divisive Methods



(b) Dendrogram

Figure 14.3: Hierarchical Clustering

The quality of this clustering depends on the merge / split decisions taken at different levels. The split and merge are decided based on similarity metrics applied on two clusters known as the (***linkage measures***). Fig. 14.4 shows the summary of various linkage measures used, where x_1, x_2 represents two objects in two different clusters C_1, C_2 respectively.

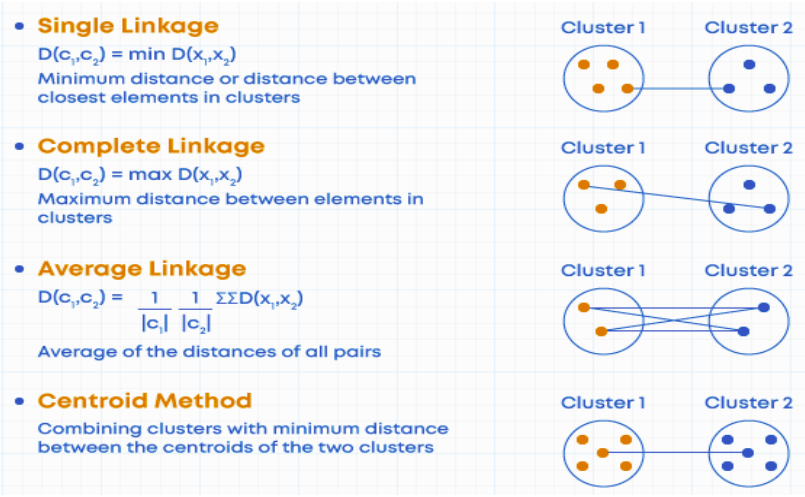


Figure 14.4: Various Linkage Measures

- Hierachical clustering methods do not scale to large datasts because each decision of merge or split needs to evaluate many pairs of clusters. The total number of pairs to evaluate is a combinatorial problem and it is computationally expensive.

14.4 Density-Based Methods

Centroid based and hierarchical based clustering methods are designed to find spherical-shaped clusters. They have difficulty finding clusters of arbitrary shape. Density based clustering helps to solve this problem by following a different approach. To find clusters of arbitrary shape, we can model clusters as dense regions separated by sparse regions as in Fig. 14.5. DBSCAN is a popular algorithm based on this idea.

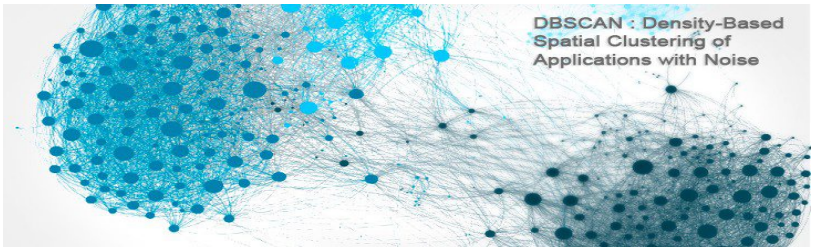


Figure 14.5: Clusters with Arbitrary Shape

How does DBSCAN quantify density?. Desnity around a point is defined using two hyper parameters.

1. ϵ : Its the radius of a neighborhood, $\epsilon > 0$. This is estimated using trial and error. For all points we compute the distance for its *MinPts* number of neihbors and using a knee plot, identify the right value for ϵ.

2. ***MinPts***: Its the minimum number of points to consider a neighborhood region as dense region. *MinPts* is set based on domain knowledge or sometimes based on rule of thumb - it should be greater than d dimesnion of data points and atleast $2d$.

Number of points within the hypersphere of radius ϵ centered at point x_i is the **density** of point x_i, if this region has a desnity greater than or equal to MinPts, its called a **dense region** otherwise its called a **sparse region**. We need to know a couple of terms used in the DBSCAN before we proceed to the algorithm. These terms are shown in Fig. 14.6.

- **Core point** : A point which has atleast MinPts points. This is the center of a dense region. Every dense region contains atleast one core point. Core points are the pillars of dense regions.

- **Border point** : A point not a border point but within ϵ distance from some core point.

- **Noise point** : A point neither a core point nor a border point. The noise points won't be part of any cluster.

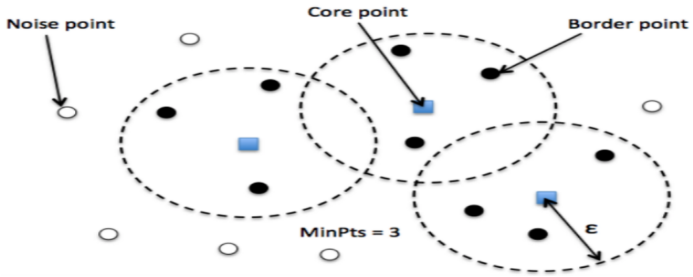


Figure 14.6: DBSCAN Terminologies

In a nutshell DBSCAN algorithms connects all core objects and their neighborhoods if its is within ϵ distance and form dense regions as clusters. We also need to define two more terms for the complete understanding of this algorithm 3.

- **Density edge** : If x_i, x_j are two core points and the distance between them is atmost ϵ, we can connect x_i, x_j by an edge called a density edge.
- **Density connected points** : Two core points x_i, xj are said to be density connected points if there exist a sequneces of density edges in the graph that forms a path from x_i to x_j.

- The worst case time complexity is $\mathcal{O}(N^2)$. Space complexity is $\mathcal{O}(N)$, d is ignored since DBSCAN is not typically used in high dimensional data.
- In contrast to k-means algorithm, we don't need to specfy the value of k, algorithm automatically finds it.
- If the dataset contains varying densities (many densities)the DBSCAN may give poor results. Also this algorithm is very sensitive to hyper-parameters.

14.5 Interpret Clustering Results & Adjust Clustering

Because clustering lacks "ground truth", it complicates to assess the quality of clustering. First, perform a visual check that the clusters look as expected, and that examples that we consider similar do appear in the same cluster. Here are guidelines that we can iteratively apply to improve the quality of clustering. We can check the common metrics for assessing the quality of clustering.

Algorithm 3 DBSCAN

Input

D: Data set containing N objects,

ϵ: Radius parameter

MinPts: the neighborhood density threshold

Output:

A set of density-based clusters.

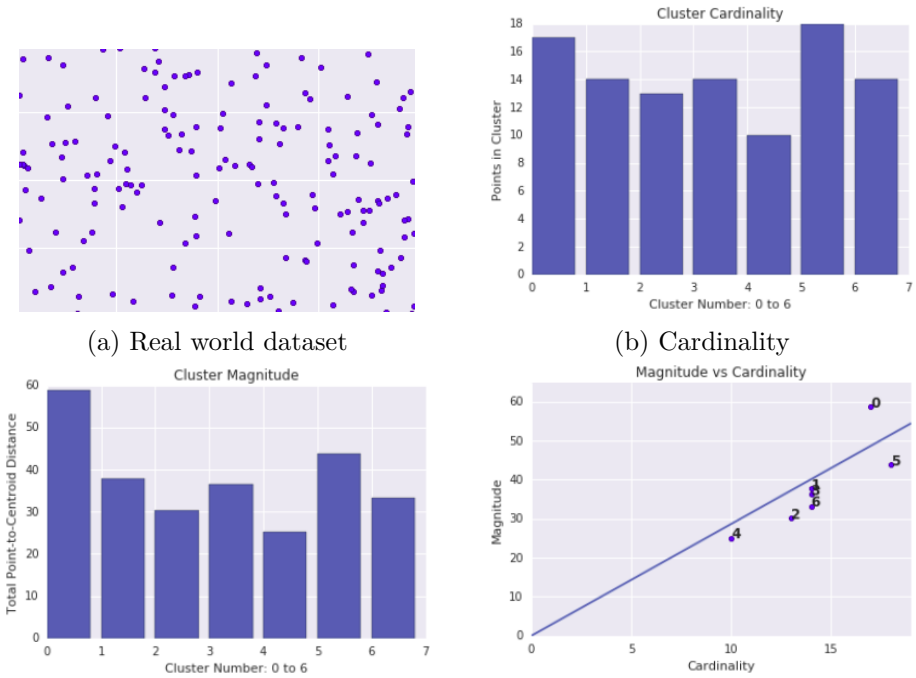
1: Label all the points in the dataset as any of the Core, Border or Noise using ϵ & *MinPts*.
2: Remove all noise points. The noise points will be on the sparse regions.
3: **for all** core points x not added into any cluster **do**
4: Add x to the cluster.
5: Add all density connected points of x into the cluster.
6: **end for**
7: **for all** border points b **do**
8: assign it to the cluster of its closest core point.
9: **end for**

1. **Cluster cardinality** is the number of examples per cluster. Plot the cluster cardinality for all clusters and investigate clusters that are major outliers.

2. **Cluster magnitude** Cluster magnitude is the sum of distances from all examples to the centroid of the cluster. Similar to cardinality, check how the magnitude varies across the clusters, and investigate anomalies.

3. **Magnitude vs. Cardinality** Notice that a higher cluster cardinality tends to result in a higher cluster magnitude, which intuitively makes sense. Clusters are anomalous when cardinality doesn't correlate with magnitude relative to the other clusters. Find anomalous clusters by plotting magnitude against cardinality.

4. **Performance of downstream system** : Since clustering output is often used in downstream ML systems, check if the downstream system's performance improves when your clustering process changes. The impact on your downstream performance provides a real-world test for the quality of your clustering but the downside is that this cheque is costly to perform.



(a) Real world dataset

(b) Cardinality

(c) Magnitude

(d) Cardinality vs Magnitude

Figure 14.7: Clustering Checklist

There are some questions to ask when the above problems are noticed.

- Is your data scaled?
- Is your similarity measure correct? clustering algorithm is only as good as the similarity measure.
  - Make sure your similarity measure returns sensible results. The simplest check is to identify pairs of examples that are known to be more or less similar than other pairs.
  - Ensure that the similarity measure for more similar examples is higher than the similarity measure for less similar examples.
  - If you find examples with inaccurate similarities, then your similarity measure probably is not a good fit.
- Is your algorithm performing semantically meaningful operations on the data?
- Is the hyperparam tuning done?

14.6 Measuring the Quality of Clustering

How well the data points in the clusteres are grouped? Is there anyway to quantify these groupings done by clustering. Yes - we have three metrics to quantify the groupings such as,

- **Inertia or Distortion**
- **dunn-index**
- **Silhouette coefficient**

Inertia measures how well a dataset was clustered by k-means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster (magnitude) and adding up all such magnitudes. Plot this inertia against the number of clusters k and we can use elbow method to find optimal value for k as shown in Fig. 14.8.

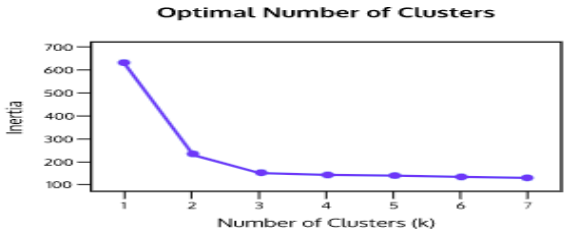


Figure 14.8: Optimal Value of $k = 3$ using Inertia

- There is a trade off between increase in k and decrease in inertia. As k increases, clusters become smaller, and the inertia decreases.
- The reduction in loss becomes marginal with increasing in k. Mathematically, that's roughly the where the slope ($\tan(\theta)$) crosses above -1 ($> 135°$).

The **Dunn index** aims at quantifying the compactness of clustering. A cluster is compact if there is only small variance between members of the cluster. The *dunn*_index concept is shown in Fig. 14.9 and is measured using Eq. 14.1, where i, j index enumerate for the intercluster distance and l, m enumerate for intracluster distance.

$$dunn\_\text{index} = \frac{\min_{i,j} d(x_i, x_j)}{\max_{l,m} d(x_l, x_m)} \tag{14.1}$$

- The numerator of *dunn*_index measures the cluster separation and denominator measures the cluster variance.

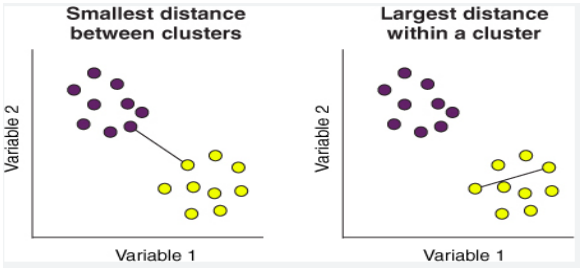


Figure 14.9: Dunn Index

- Higher the *dunn*_index, higher the chances of clusters being separated well and more compact.

Another intrinsic metric to evaluate the quality of a clustering is **silhouette analysis**. To calculate the **silhouette coefficient** of a single sample x_i in our dataset, we can follow the below three steps. This concept is ilustrated in Fig. 14.10.

1. Calculate the **cluster cohesion** $a_\mu^{(i)}$ as the average distance between a sample x_i and all other points in the same cluster.
2. Calculate the **cluster separation** $b_\mu^{(i)}$ from the next closest cluster as the average distance between the sample x_i and all samples in the next closest cluster.
3. Calculate the silhouette $s^{(i)}$ as the difference between cluster cohesion and separation divided by the greater of the two, as shown below:

$$s^{(i)} = \frac{b_\mu^{(i)} - a_\mu^{(i)}}{\max\left(b_\mu^{(i)}, a_\mu^{(i)}\right)} \tag{14.2}$$

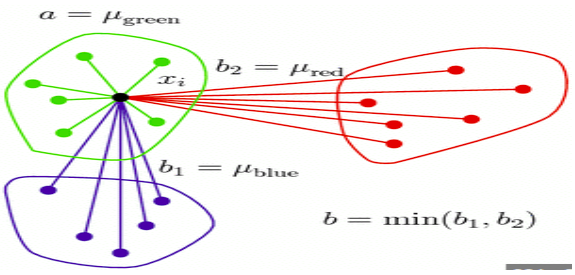


Figure 14.10: Silhouette Coefficient

- The silhouette coefficient is bounded in the range -1 to 1, we get close to an ideal silhouette coefficient of 1 if $b \gg a$.

15

Neural Networks

Have you ever wondered, how your brain recognizes numbers? No matter how the digits or numbers looks like, brain will relate that to the best possible pattern and concludes the result. This is where the thinking came out to make something which can recognize similar number patterns, and that is where Neural Networks starts.

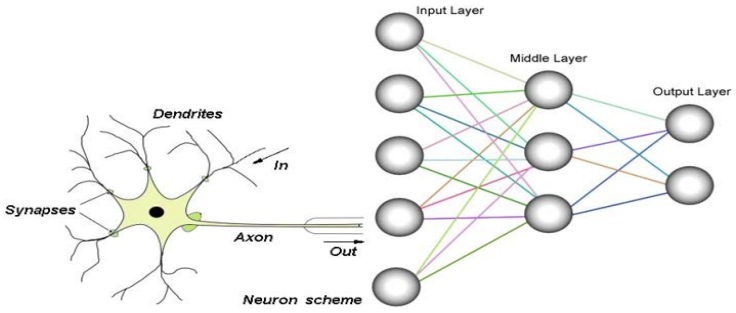


Figure 15.1: Neural Network - Biological vs Artificial

- A Neural Network is a system designed to operate like a human brain. Human information processing takes place through the interaction of many billions of neurons connected to each other sending signals to other neurons.
- Human brain consists of billions of neurons and trillions of interaction. Its a massively parallel processing system. Modern computers have processing units which is faster than the human brain neurons but the number of such units are limitted to a few hundreds or a few thousands.

A technical neural network consists of simple *processing units* called **neurons**, directed and weighted connections between those neurons (weighted graph). Fig. 15.1 shows a biological neural network and an artificial neural network.

The neurons are grouped in multiple layers: One **input layer**, a few **hidden layers** (middle layers) and one **output layer**. The signals pasess from input to output layer in the forward direction only. So it's called multilayered feed forwared neural network. If edges exist between all pairs of neurons in all adjacent layers then its called **Fully connected feed forward neural network**.

- The number of inputs to the network and the number of outputs from the network are defined by the problem specifications (d neurons in input layer for d features, m neurons in output layer for m outputs). The number of hidden layers and number of neurons in the hidden layers are hyperparameters and needs to be tuned.
- Each edge has a weight that get multiplied by the output from neurons in previous layer and will be passed to neuron in the next layer. The neurons can have a bias value which is not multiplied to output from previous layer.
- A net input of a neuron is the sum of output signal from previous layer and the bias. As the bias is not multiplied with input, it can ensure that the net input is non-zero even if the input is zero. Thus bias helps neuron in activation.

To understand why we need to use multi layered feed forward neural networks, lets take the examples shown in Fig. 15.2.

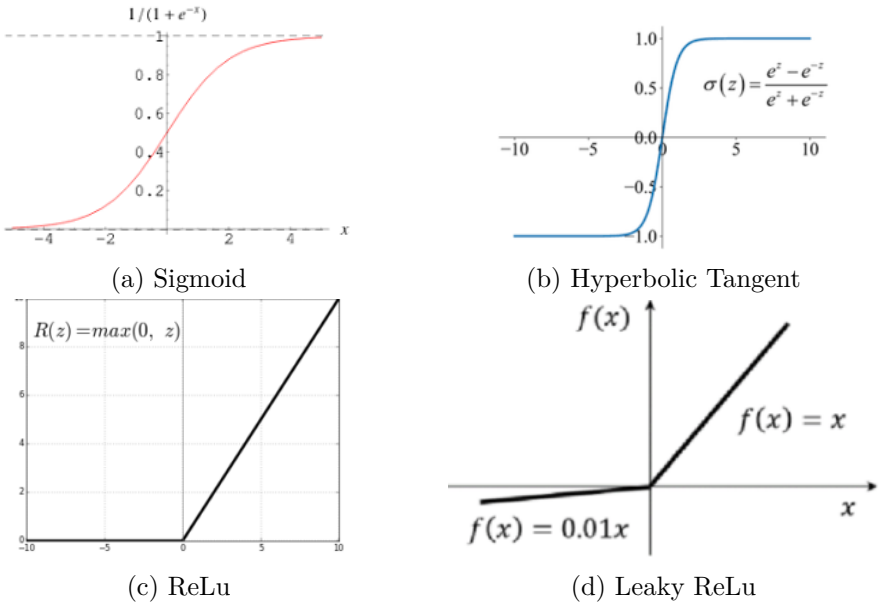
(a) Separable with $(x_1 * x_2)$ (b) Non Separable

Figure 15.2: Linear Separability of Data points with Feature Crossover

In real world cases the data are often the type showin in Fig. 15.2b. Adding more layers doesn't help in creating a non-linear separable decision boundary because multiplying linear equation $(W_1.X)$ with another weight (W_2) will result in another linear equation only.

- To create a non-linear decision boundary we need to introduce non-linearity in the activation of neurons. i.e A non linear activation function must be used.
- By stacking nonlinearities on nonlinearities will let the model learn very complicated relationships between the inputs and the outputs.

Popular activation functions used for neurons are given in Fig. 15.3.



(a) Sigmoid (b) Hyperbolic Tangent

(c) ReLu (d) Leaky ReLu

Figure 15.3: Activation Functions

- **Sigmoid** ($\sigma(z)$) : Small change in input value causes small change in output value.
- **Tanh** : Very similar to sigmoid function. Tanh(z) = 2*$\sigma(z)$) - 1
- **ReLu** : Its a popular activation function used in neural network especially in deep neural networks.
- **Leaky ReLU** : A variant of ReLu. As the output value of ReLU is 0 for negativie input values, the activation will be 0 and this problem is known as **dead activation**, leaky ReLU over come this problem by putting a +ve slope for negative values.

15.1 Weight Initialization

Before we train the neural network, we need to initialize the weights. Can we initialize all weights with 0s like we initialize many variables? **No**!. We cannot initialize all weights with 0s or same values, if we do so, the gradients computed would be 0 or same values and it doesn't support learning. For the learning algorithm to work well, the weight initialization should have enough variance like weights taken from normal distribution. Many weight initialization methods have been proposed based on **fan-in** & **fan-out** values

of a neuron which represents the number of incoming edges and outgoing edges of a neuron.

- **Uniform initialization** : Here weights are initialized using the uniform distribution
  - $\left[\frac{-1}{\sqrt{fan-in}}, \frac{1}{\sqrt{fan-in}}\right]$.
- **Xavier or Glorot initialization** : There are two methods of Xavier initialization.
  - $N(0, \sigma)$ initialization where $\sigma = \frac{2}{fan-in+fan-out+1}$
  - Uniform initialization from $\left[\frac{-\sqrt{6}}{\sqrt{fan-in+fan-out}}, \frac{\sqrt{6}}{\sqrt{fan-in+fan-out}}\right]$.
- **He Initialization** : This works fairly well for ReLU activation.
  - $N(0, \sigma)$ initialization where $\sigma = \sqrt{\frac{2}{fan-in}}$
  - Uniform initialization from $\left[-\sqrt{\frac{6}{fan-in}}, \sqrt{\frac{6}{fan-in}}\right]$.

15.2 Backpropagation Learning

Backpropagation is a gradient descent ("**steepest descent algorithm**") based training algorithm for MLP. Weights and biases are the parameters of MLP and we find the best parameters that minimizes the loss (L). Output layer is the only layer for which the loss is directly defined. In backpropagation we are minimizing the loss by adjusting the weights. We first do a forward pass and get the predicted value ($\hat{y}$) and we can compare it with the ground truth (y) to get the loss. The **cost function** is the mean of individual losess.

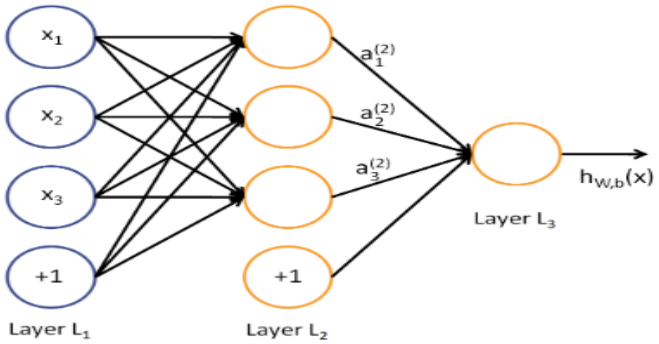


Figure 15.4: Backpropagation Training

Let $w_{ij}^{(k)}$ denote the weight of edge that connects i^{th} neuron to j^{th} neuron in next downstream layer k and $b_j^{(k)}$ the bias for j^{th} neuron. Backpropagation

algorithm assumes the loss L is proportinal to the weight $w_{ij}^{(k)}$, so it computes the gradient of L with respect to $w_{ij}^{(k)}$ takes a fraction of this gradient using **learning rate** $(0 < \alpha < 1.0)$ and update the parameters.

- Gradient for a weight/bias needs to be computed based on the chain of dependency of loss L on the weights/biases. So backpropagation uses chain rule of differentiation for computing gradients.

First we look into the case where k is the output layer. The weight change for a weight $(w_{ij}^{(k)})$ connecting a node i in last hidden layer $k-1$ to a node j in output layer k can be written as Eq. 15.1, where the negative sign indicates weight changes are in the direction of decrease in loss.

$$\Delta w_{ij}^{(k)} = -\alpha \frac{\partial L}{\partial w_{ij}^{(k)}} \tag{15.1}$$

But the loss indirectly depends on many other variables at node j - activation denoted as a_j^k, activation inturn depends on the netnput at z_j^k, netinput depends on so many other weights from upstream layer and the bias $b_j^{(k)}$. So we need to use chain rule of differentiation to find the gradients.

$$\frac{\partial L}{\partial w_{ij}^{(k)}} = \underbrace{\frac{\partial L}{\partial a_j^{(k)}} \frac{\partial a_j^{(k)}}{\partial z_j^{(k)}}}_{\text{error - } \delta_j^{(k)}} \frac{\partial z_j^{(k)}}{\partial w_{ij}^{(k)}} \tag{15.2}$$

First factor in Eq. 15.2 is the partial derivative of loss function, second factor is the derivative of activation function. So the partial derivative becomes

$$\frac{\partial z_j^{(k)}}{\partial w_{ij}^{(k)}} = \frac{\partial \Big(\sum a_i^{(k-1)} * w_{ij}^{(k)} \Big)}{\partial w_{ij}^{(k)}} = a_i^{(k-1)} \tag{15.3}$$

We will find it notationally convenient to treat the first two factors as a single quantity, an "error" term (δ_j) at the downstream node. Using the above partial derivatives we can, conveniently write the equation as

$$\frac{\partial L}{\partial w_{ij}^{(k)}} = \delta_j^{(k)} * a_i^{(k-1)} \tag{15.4}$$

In the case where k is a *hidden layer*, $\frac{\partial L}{\partial a_j^{(k)}}$ is not quite simple. We need to check the influence of $a_j^{(k)}$ to the loss through propogation to next layer.

Let l denote the index of neurons in the next layer $(k+1)$. This leads to another chain rule

$$\frac{\partial L}{\partial a_j^{(k)}} = \sum_{l \in (k+1)} \underbrace{\frac{\partial L}{\partial a_l^{(k+1)}} \frac{\partial a_l^{(k+1)}}{\partial z_l^{(k+1)}}}_{\text{error - } \delta_l^{(k+1)}} \frac{\partial z_l^{(k+1)}}{\partial a_j^{(k)}} \tag{15.5}$$

The second factor is the derivative of net input with respect to activation.

$$\frac{\partial z_l^{(k+1)}}{\partial a_j^{(k)}} = w_{jl}^{(k)} \tag{15.6}$$

Substituting in Eq 15.6 we get

$$\frac{\partial L}{\partial a_j^{(k)}} = \sum_{l \in (k+1)} \delta_l^{(k+1)} * w_{jl}^{(k)} \tag{15.7}$$

Putting all together, the partial derivative term for hidden layer neuron becomes

$$\frac{\partial L}{\partial w_{ij}^{(k)}} = \underbrace{\Big(\sum_{l \in (k+1)} \delta_l^{(k+1)} * w_{jl}^{(k)} \Big) * \Big(\frac{\partial a_j^{(k)}}{\partial z_j^{(k)}} \Big)}_{\delta_j^{(k)}} * \Big(\frac{\partial z_j^{(k)}}{\partial w_{ij}^{(k)}} \Big) \tag{15.8}$$

Even though this formula differs from the output-layer case Eq. 15.2, we can write a single formula by employing our definition of the error term (δ_j). So the equation becomes

$$\frac{\partial L}{\partial w_{ij}^{(k)}} == \delta_j^{(k)} a_i^{(k-1)} \tag{15.9}$$

We've just buried the differences in the equations for the output and hidden layers in the error term (δ_j). This notation simplifies the implementation of backpropagation algorithm as mentioned below,

- We need to write just single function which computes the weight change of an edge, given the activation of the upstream node and error term in the downstream node.
- The error term varies based on whether the downstram node is a hidden layer node or output layer node.
- The compution of error term in hidden layer requires the error term in output layer first. Thus, computation of the error terms must proceed

backwards through the network, beginning with the output layer and terminating with the first hidden layer (hence the name back propagation).

The above steps can be efficiently implemented using the idea of **Dynamic Programming** where it reuses the derivatives in calculating weight updates. Backpropagation algorithm is given in Algorithm 4

Algorithm 4 Backpropagation

Input

D: a data set containing N objects,

α: the learning parameter

Output:

A set of learned weights W.

1: Initialie all weights W.
2: **repeat**
3: **for all** X_i in D: **do**
4: Pass X_i to the network, is called **forward propagation**.
5: Compute the loss using $\hat{y_i}, y_i$.
6: Compute all gradients using chain rule and dynamic programming idea.
7: Update the weights using the gradients computed as follows,

$$W_{t+1} = W_t - \alpha * \Big[\frac{\partial L}{\partial W}\Big]_t$$

8: **end for**
9: **until** the weigths converged or error is very low

There are three popular modes to feed the dataset for neural network training as given in Table **??** and the effect of learning is compared in Fig. 15.5.

15.3 Softmax Classifier

How do we leverage the neural network for multiple clasess? One vs Rest is a popular method for extending to multiple clasess. In this method we use one binary classifier for each class but its computationally expensive when we have more clasess.

Table 15.1: Different Modes of Training Backpropagation

Gradient Descend	**Stochastic Gradient Descend**	**Mini-batch Gradient Descend**
Entire data feed is used to compute gradient and update the weights	Gradient is computed per observation	Gradient is computed using a batch of observations
Suitable when data feed size is small	Suitable when data feed size is huge	Suitable when data feed size is high
Gradient is least noisy, least sensitive to outliers and fastest to converge	Gradient is highest noisy, highly sensitive to outliers and slowest to converge	Gradient is less noisy, less sensitive to outliers and converges faster
Weight update is done for entire data set so its computationally cheap	weight update is done per observation, so its the most computationally expensive	weight update is done per batch

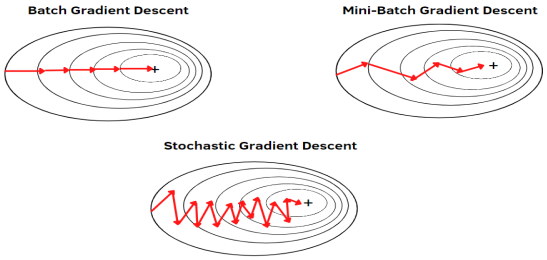


Figure 15.5: 3 Modes of Training

Softmax extends the neural network for multiple clasess. Softmax layer in neural network has same number of neurons as that of the number of output classess. Each neuron in the softmax layer predicts the probability that an input belong to a particular class.

- Softmax layer takes net input and convert that into a probability distribution. This additional constraint helps training converge more quickly than it otherwise would.
- Softmax is implemented through a neural network layer just before the output layer. Softmax is not computationally expensive compared with One vs Rest method.
- Softmax assumes that each example is a member of exactly one class. For multilabelled classification we have to rely on other methods like one logistic regression for one class or make the output layer with multiple neurons one for each class.

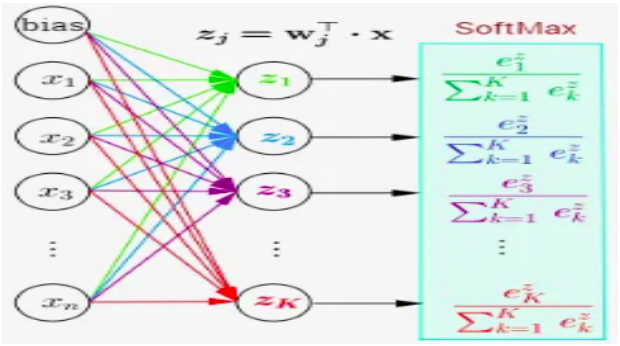


Figure 15.6: Softmax Classifier

Softmax is fairly cheap when the number of classes is small but becomes prohibitively expensive when the number of classes climbs. **Candidate sampling** is a technique to reduce the computation by calculating the probability for all the positive labels but only for a random sample of negative labels.

15.4 Optimizers for Training

Backpropagation algorithm is based on gradient descent, i.e at every step of the iteration it computes the gradient of loss with respect to weight and update the weight by taking fraction of the negative of the gradient. But one problem with this approach is that the gradient will be close to zero with gentle slope of the loss curve and gentle slope is quiet common in most of the loss surfaces for neural networks.

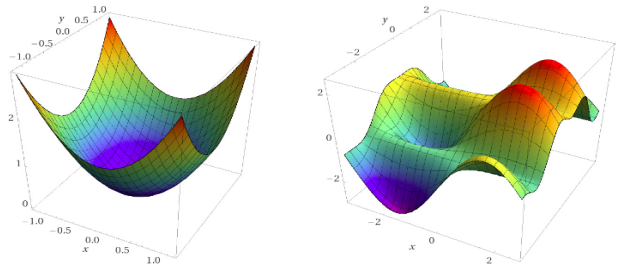


Figure 15.7: Loss Surfaces

In general there are two types of loss surfaces due to two types of loss functions,

- **Convex loss functions** : These functions are convex in shape and it has only one maximum or minimum. In other words the maxima or minima that we get for these type of functions are always global maxima, global minima. The examples for convex loss functions are:

Logistic loss (logistic regression), linear loss (linear regression), hinge loss (SVM model), mean squared error.

- **Non Convex loss functions** : For these type of functions we have local and global minima, maxima and saddle points. eg: MLP loss function.

15.4.1 Momentum

Imagine a ball is rolling down a hill and get stuck in a pit. If the ball had enough momentum it will be able to come out of the pit by accelerating the direction towards the valley area. **Momentum** is a technique that helps this acceleration. It also improve the slow convergence of gradient descend on gentle slopes and also might help to escape from local minima.

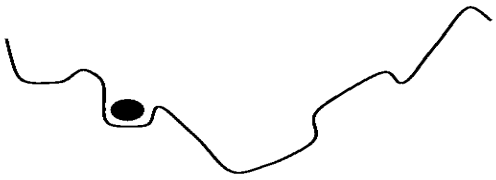


Figure 15.8: Local Minima

- The basic idea of momentum is that we also account previous gradients and if many of such previous gradients points in one particular direction we accelarate towards that direction.

Momentum can be incorporated into the weight update formula by forming a recurrence equation, as a result we are taking the exponential weightages of previous gradients. This is given in equation 15.10 where $\eta \in (0.0, 1.0)$ represents the fraction for the weighted gradients, y_k is the activation of upstream neuron, δ_j is the error of downstream node and t denote the current iteration number.

$$\Delta W_{kj}^{(t)} = \alpha \delta_j y_k + \eta \Delta W_{kj}^{(t-1)} \tag{15.10}$$

- When the gradient keeps pointing in the same direction, momentum will increase the size of the steps taken towards the minimum and it may rush past the minimum, its therefore often necessary to reduce the global learning rate α, when using a lot of momentum η.
- On the otherhand, if the gradient keeps changing direction, momentum will smooth out the variations.
- There are advanced optimizations algorithms developed based on the momentum idea. **Adagrad, Adadelta, Adam** are three popular optimization algorithms.

15.5 Deep Neural Networks

Neural networks with more than one hidden layer is considered as **deep neural networks**. Deep learning sets the foundation for most of the natual language processing, natural language understanding works like voice assistant sytems and computer vision problems. Although the idea of neural networks were developed in early 80s, deep neural network didn't attain much popularity that time due to lack of enough data and computation power to train the model. Today we have huge amount of data (big data) through online platforms like social media and enogh computation power due to the developement of GPUs.

Making the network more deep has the benefit of learning complex patterns and helps us to build state of the art models to solve problems in speech processing, computer vision, natual language processing, robotics etc. But deep neural networks put challenges in backpropagation training algorithm.

- Backpropagation algorithm uses chain rule of differentiations that multiplies lot of gradients values together. If the gradient values are too small, it can lead to numeric underflow and no update to weights.
- As we make the network more deep, the number of parametes to learn also increases a lot and this put the risk of overfitting.
- Due to large number of layers in deep neural networks, if we feed similar batch of inputs, it may happen that the network layer sees it as two entirely different inputs.

15.5.1 Vanishing Gradient

Sigmoid function has some limitations, the value of sigmoid function get approximated to 1 if the net input is greater than a small value (≥ 5) and value get approximated to 0 if the net input is small negative values (values ≤ -5). This make the derivative to 0 for values greater than a small positve number or less than a small negative number. The derivative of sigmoid can be expressed as the product of sigmoid function itself.

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z)) \tag{15.11}$$

So in the chain rule of differentialtions, if we use this derivative value the weight update will be 0 and this will stop the training process. As Tanh(z) properties are also similar to sigmoid function, this function also has the same problem of vanishing gradient. So what is the solution? **ReLu!!**

- ReLU derivative has only two values $\{0, 1\}$ so its simple to compute and ReLu helps the algorithm to converge faster. Leaky ReLu is another option as it has similar properties as that of ReLu.

15.5.2 Dropout and Regularization

Another important problem that we face with deep neural netwroks is the overfitting problem. Like other models L1 and L2 regularization can be applied to MLP to control bias variance trade off. But due to the large number of parameters this method is seldom used.

The overfitting is casued by learning too many parameters, what if we drop some of those parametes? This technique is know as the **Dropout method**. This is more elegant way to control the overfitting and also very chep to implement when compared with L1, L2 regularization. At every iteration of backward pass in backpropagation we drop some of the weights of deep layers from updating. This is done by setting a **dropout rate** $\in (0, 1)$ for deep layers.

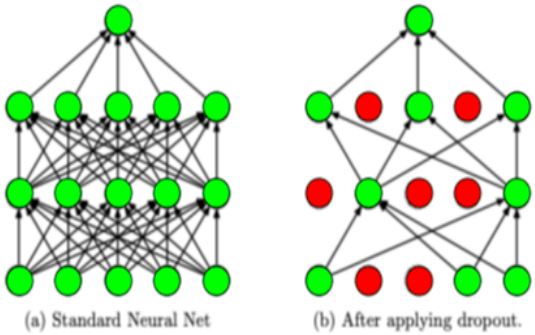


Figure 15.9: Dropout Method

- We can think the dropout method concept as analogus to Random Forest column sampling method where we randomly select a few columns to build individual tree.
- We set dropout rate high when we overfit on small data set and low dropout rate otherwise.
- At forward pass step, all the edges should be present, but we multiply the weights by dropout rate to adjust the drop out done during weight update.

15.5.3 Batch Normalization

As deep neural networks has many layers, the net input of deeper layers is a composite function of input value, that is it consists of many multiplications and additions. This can make the net input value of deep layers to be a much higher positive or negative number when compared with input value (input value is usually scaled value with $\mu = 0$ & $\sigma = 1$). This makes the μ & σ of deeper layers to be way different from the input. This problem is

known as the **internal covariance shift**. Having large net inputs in the deeper layer creates problem in training

- It increases the number of iterations to converge and also large negative values makes the ReLu to go in dead activation state.
- When we feed slightly different batches of an input (say a batch of white dog images instead of batch of black dog images), deep layer shows significant difference in net inputs though both batches are images of dogs.

How do we get rid of this problem ? The solution is **batch normalization**. Batch normalization method adds a network layer that performs the scaling operation to the net input value. As this layer does the scaling on net inputs, it supports the regularization and helps to reduce overfitting. This layer is added after the net input is calculated but before the activation function is applied. Batch normalization is pretty similar to column normalization but with an additional operation of rescaling and offsetting. Let M be the mini-batch size of batch b and a net input z_i will be rescaled to $\hat{z}_i$ in batch normalization by following the four steps below :

1. $\mu_b \leftarrow \frac{1}{M}\sum_{i=1}^{M} z_i$
2. $\sigma_b^2 \leftarrow \frac{1}{M}\sum_{i=1}^{M}(z_i - \mu_b)^2$
3. $\hat{z}_i \leftarrow \frac{z_i - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}}$, ϵ is a small positive number added to avoid divison by zero error.
4. $\hat{z}_i \leftarrow \gamma \hat{z}_i + \beta$, this is the rescale and offset step. Two hyperparameters β and γ can be learned with backpropagation algorithm.

15.5.4 Putting it all together : training deep MLP steps

- **Preprocess** : data normalization before feeding into network.
- **Hyperparameters** : architecture of neural network. Number of layers and number of neurons.
- **Activation Function** : if network is shallow tanh/sigmoid if deep use ReLU
- **Weight Initialization** : initialize the network with standard methods.
  - Xavier / Glorott : Either uniform or normal if the activation functions are sigmoid/ tanh
  - He : Either uniform or normal if the activation function is ReLU

  - Gaussian distribution with small variance

- **Batch Normalization** : if network is deep use batch normalization in deep layers.
- **Dropout** : use dropout for regularization. Dropout rate is a hyperparmaeter and it should be chosen based on cross validation.
- **Optimizers** : Adam is the most commonly used over all the other methods.
- **Loss Function** : Use log loss for two class problem, cross entropy for multiclass and Squared loss for regression problems.
- **Gradient Monitoring** : Apply things like gradient clipping if needed.
- **Plots** : Plot the train loss, validation loss against the number of epochs. If overfit happens rework on tuning dropout method & batch normalization methods.

15.5.5 Embeddings

Suppose we want to find similar movies from a list of movies. There are different aspects from a movie can be looked into for similarity. We can find similarity based on the genre of movie, the decasde in which the movie was released, movie is arthouse or blockbuster, the age group for which the movie is meant for. In general we have many dimensions to look for the similarity. There are real world cases where we can have thousands of dimensions. Training neural network models on such huge dimensional data would be inefficient for two reasons,

- **Amount of data** The more weights in your model, the more data you need to train effectively. Also the model tend to overfit with increase in the number of parameters
- **Amount of computation** The more weights, the more computation required to train and use the model. It's easy to exceed the capabilities of your hardware.

What is the solution ? **Embeddings !!**. An embedding is a relatively *low-dimensional space* into which you can translate high-dimensional vectors. Embeddings make it easier to do machine learning on large inputs like sparse vectors representing words. Ideally, an embedding captures some of the semantics of the input by placing *semantically similar inputs close together* in the embedding space. An embedding can be learned and reused across models.

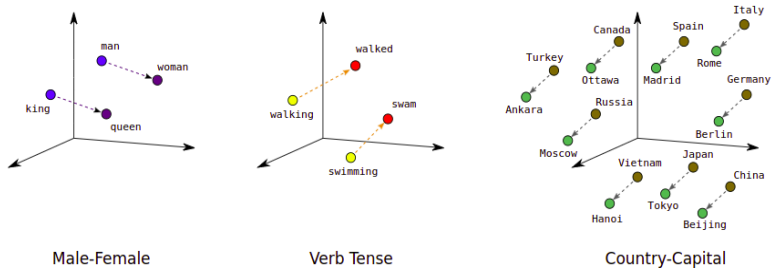


Figure 15.10: Embedding

- We can convert sparse vectors in high dimension to dense vectors in lower dimensional embedding. The position (distance and direction) in the embedding vector space can encode semantics of words.

- This sort of meaningful space gives machine learning system opportunities to detect patterns that may help with the learning task.

An embedding is a matrix in which each column is the vector that corresponds to an item in your vocabulary. To get the dense vector for a single vocabulary item, you retrieve the column corresponding to that item. But how would you translate a sparse bag of words vector or a text of words?

- We could retrieve the embedding for each individual item and then add them together. If the sparse vector contains counts of the vocabulary items, we could multiply each embedding by the count of its corresponding item before adding it to the sum.

- Each dimension in the embedding space represents **latent dimension**, its called latent because it doesn't exist in the original data but infered it.

- One downside of this method is the interpretability, we don't have one to one mapping from input dimension to embedding dimension. Sometimes, we can look at the embeddings and assign semantic meanings to the dimensions, and other times we cannot.

15.5.6 Auto Encoders

An **autoencoder** is an unsupervised neural network designed for extracting the latent dimensions. This algorithm sets the input and output be same (hence known as autoencoder) but has reduced number of neurons in the hidden layer to learn latent dimensions. Auto enocders find application in image compression, image denoising etc.

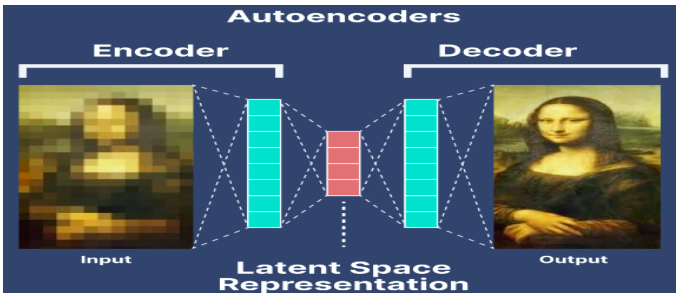


Figure 15.11: Autoencoder

- By setting the target values same as that of the input and having the reduced number of neurons in the hidden layers, the model is forced to learn important structure in the input and denosie the input.
- Autoencoders are trained using backpropagation algorithm. RMSE is used as the loss function. If the RMSE is low, model has learned good representation of the input.
- The part between the input and embedding layer is known as the **encoder** part and part between embedding and and output layer is known as the **decoder** part. In the decoder side we try to *reconstruct* the original input from lower dimension.
- If there are correlations among input features the model would be able to utilize that to reduce the dimension. On the other hand if all input features are identically and independently distributed then its hard for the model to reduce dimension.

15.5.7 Word2Vec : A word embedding technique

Word2vec is an algorithm invented at Google for training word embeddings i,e mapping each word in the corpus to a vector in the embedding space. This mapping is done by considering semantically similar words get mapped to geometrically close embedding vectors, for eg. The word “tea” and “coffee” frequently appear close to the word “beverages” and this fact reflects their semantic similarity.

As the linguist John Firth put it in 1957, "You shall know a word by the company it keeps". Word2Vec exploits *contextual information* like this by training a neural net to distinguish actually **co-occurring groups** of words from randomly grouped words. Word2Vec exploit this idea and can learn **“KING - MAN + WOMAN = QUEEN”**, kind of relationships that exist betwenn words in real life.

Lets take a sentence to explain the core idea of word2vec. “The quick brown fox jumps over the lazy dog” in this sentence, we first form **focused**

word and **context word** pairs. Thease pairs are formed by using a window of fixed size say 2. (typically size of 5 to 10 is used). The Fig. 15.12 below shows the focused word "fox" and the context words {quick, brown, jumps, over}. The context and focused word pairs helps to learn the semantics of words.

Figure 15.12: Focused word & Context words

We first represent all the words in the vocabulary of size V as V-dimensional one-hot encoded vectors this includes the focus word and context words also represented as one-hot encoded vectors. Then we train a shallow neural network with just one hidden layer of much lower dimension $d << V$ (typicaly $d = 300$) with linear activations and fully connected network architecture. We also have a softmax layer in the output layer to predict words. There are two ways to learn the embeddings as shown in Fig. 15.13.

- **Continuous bag of words (CBOW)** : Here we set the context words in the input side and train the model to predict the focus word. Its used with dataset with small vocabulary size and so trains faster. With k context words, we have $k * V$ neurons in the input layer and V neurons in the output layer
- **Continuous Skip Grams** : Here we set the focus word in the input side and train the model to predict the context words. Its used for large corpus with lot of words and bit computationally expensive. With k context words, we have V neurons in the input layer and $k*V$ neurons in the output layer with k softmax units. Due to multiple softmax units this is computationally expensive.

How do we get the embedding from the model? We can extract it from the weight matrix between the input and the hidden layer. Between hidden layer and output layer we have a weight matrix of size $d \times V$ and we can pick the i^{th} column representation for i^{th} word.

15.6 Application and Cases

- Two hyperparameters of MLP are number of layers, λ the regularization term coefficient.
- As we increase the value of λ the variance decreases.

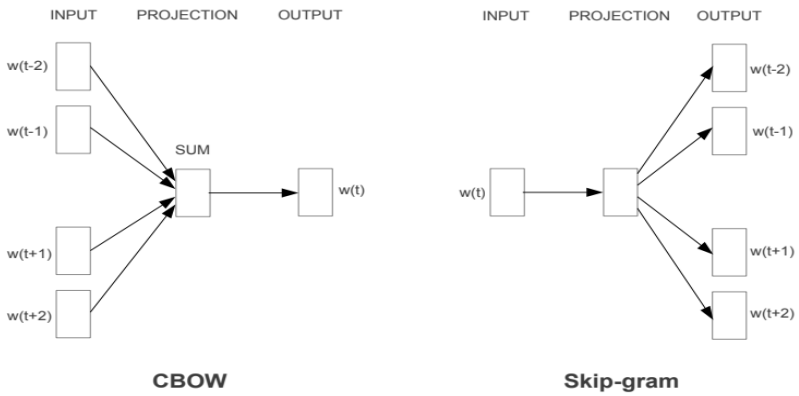


Figure 15.13: Word2Vec Architectures

16

Convolution Neural Network

In 1981 Heb a scientist did some research on cats brain which lead to the devolopement of Convolutional neural networks (**Conv Nets or CNNs**). There is one area in brain called **visual cortex** which has neurons that fires when it see objects. The primary visual cortex does the edge detection. Visual cortex is arragned in many layers like MLP, typically the shallow layers detect edges, these output are combined in subsequent layers. The idea of CNN is inspired from this brain arrangement.

CNNs are specialized kind of neural networks for processing grid-like data. eg: time-series data a (1D-grid), gray scale image data (2D-grid), colour image (3D-grid).

- CNN employs a linear operation called **convolution** which is a specialized operation for extracting features from images or grid like input.
- CNN can detect edges, objects, colour, motion etc from an image or grid like input.

CNN based applications are used in various fields like Face detection, Image classification, Face recognition etc. CNN model based applications can automatically

- Tag photos with labels like “beach”, “pets”, home etc , and no human intervention is needed
- Search for images based on the give keyword. eg: If we search for party images the model can retrieve it easily and we no longer need to search it manually

Before CNN was developed ML engineered relied on extracting pixel based features, colour histograms etc. The downside of this method is that the feature engineering part was a burden.

A CNN model takes just the image's raw pixel data as input and "learns" how to extract these features, and ultimately infer what object they constitute. CNN based model has four major stages and we will discuss each of these satages in the following sections. A typical CNN network is shown in Fig. 16.1. This network is a CNN classifer for classiying an image contains a cat or not.

1. Convolution - linear operation
2. ReLu - Non-linear function
3. Pooling - Linear opeartion
4. Fully connected Layers - Non-linear operation

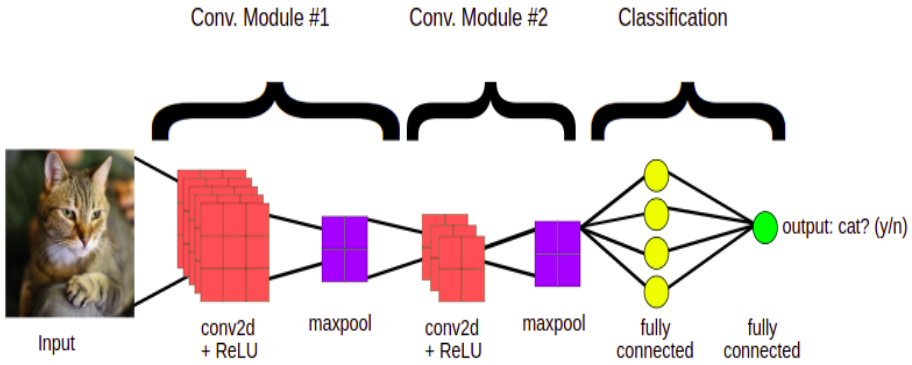


Figure 16.1: CNN Classifier

16.1 Convolution Operation

A convolution opearion extracts tiles of the **input feature map**, and applies **filters** or **kernels** to them to compute new features, producing an **output feature map**, or **convolved feature** (which may have a different size and depth than the input feature map). Convolutions are defined by two parameters:

1. **stride size** : defines the tile size to extract from image (typically 3×3 or 5×5 pixels).
2. **number of kernels** : defines the depth of output featue map

During a convolution, the filters or kernals (tesnors) effectively slide over the input feature map's grid horizontally and vertically and extract useful

features like edges, lines and other patterns automatically. For each filter-tile pair, the CNN performs element-wise multiplication of the filter matrix and the tile matrix, and then sums all the elements of the resulting matrix to get a single value. Each of these resulting values for every filter-tile pair is then output in the convolved feature matrix. This is shown in Fig. 16.2 & Fig. 16.3.

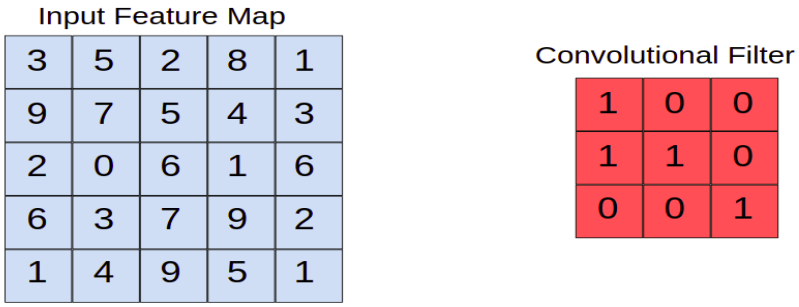


Figure 16.2: Input Feature Map & Kernel

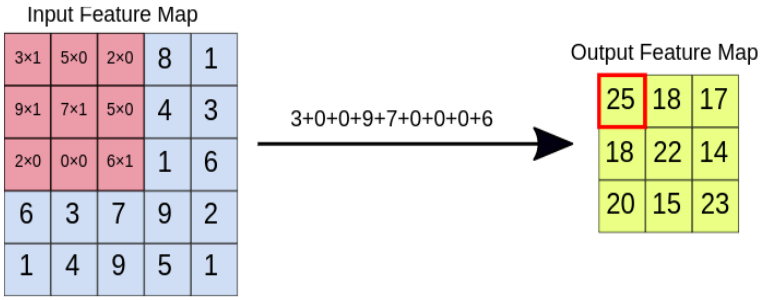


Figure 16.3: Convolution Operation

- During training, the CNN "learns" the optimal values for the filter matrices that enable it to extract meaningful features (textures, edges, shapes) from the input feature map.
- Each kernel extact one featue from the input. As the number of filters (output feature map depth) applied to the input increases, so does the number of features the CNN can extract.
- Adding more kernel increases the number of featues but the computation time too. ML engineers tries to find the best trade off between number of kernels and the exection time.
- If we use a $3D$ kernel on an RGB image, the convolved output will be $2D$ as one convolution applied on the tile generate a scalar.

- In deep ConvNets we can have multiple Convoltion layered stacked together, where shallow layer detects low level featues like edges, lines etc and deeper layer combines these low level features to form high level featues like shapes.

Fig. 16.4 shows the kernel for the horizontal edge detection and verticle edge detection. These type of kernels are called **sobel edge detectors**. The convolved output around the edge shows a high value indicating the presence of an edge.

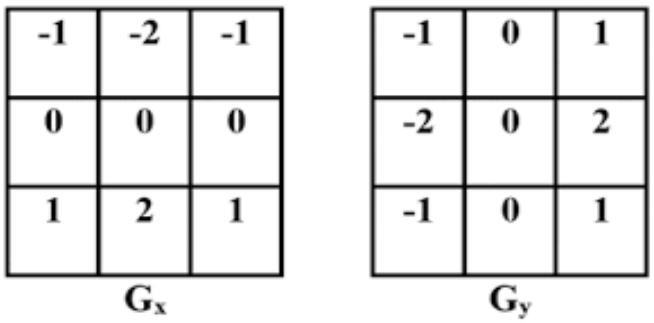


Figure 16.4: Sobel Edge Detectors

As a result of convolution operation the size of original images gets reduced. If we have image size $N \times M$ and kernel size of K, the convoled output will be of size $(N - K + 1) \times (M - K + 1)$. We can bring the image back to original size by an operation called **padding**.

- Padding is an operation where we add additional rows and columns to the original size. Usually we fill these additional columns & rows with zeros (**Zero padding**). This padding needs to be done before convolution opearation.

If we apply P-padding, then we are effectively adding $2P$ rows (top as well as bottom) and $2p$ columns (left as well as right) to the image. So post padding operation the image size becomes $(N-K+1+2P)\times(M-K+1+2P)$.

If we move the kernel S units instead of the default value 1, then its S unit **striding** opearion. Stridinghelps to reduce the size of matrix. Post striding operation the image size becomes $(\frac{N-K+2P}{S} + 1) \times (\frac{N-K+2P}{S} + 1)$.

16.1.1 Convolution - Motivation

Convolution leverages three important ideas that can help improve a machine learning system

1. **Sparse interactions**

- Convolution provides a means for working with variable input size. As we convolve over the input, this allows us to work with variable size of the input.
- CNN, typically have **sparse interactions** or **sparse connectivity** or **sparse weights**. This is accomplished by making the kernel much smaller than the input.
  - Since kernel size is very small compared to image size (thousands or millions of pixels), we can detect small, meaningful features such as edges and also it reduces the memory requirements.
  - In contrast to traditional MLP model where we have N input and M output neurons, then matrix multiplication requires $N \times M$ parameters $\mathcal{O}(N \times M)$, where as the CNN model with K kernels requires only $\mathcal{O}(K \times M)$.

2. **Parameter sharing**

   - Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter for each input interaction.
   - In contrast to MLP where every output unit interacts with every input unit and have separate parameters, in CNN we share the weights of kernel with every output unit.

3. **Equivariant representations**

   - In a deep CNN, units in the deeper layers may indirectly interact with a larger portion of the input, this allows the network to efficiently describe complicated interactions between many variables.

16.2 ReLu

Following each convolution operation, the CNN applies a **Rectified Linear Unit** (ReLU) transformation to the convolved feature, in order to introduce nonlinearity into the model. The ReLU function, $f(x) = max(x, 0)$ introduces a non-linearity and serves as an activation function for the neural network.

16.3 Pooling

After ReLU comes a **pooling step**, in which the CNN *downsamples the convolved feature* (to save on processing time), reducing the number of dimensions of the feature map, while still preserving the most critical feature

information. Pooling operates in a similar fashion to convolution. We slide over the feature map and extract tiles of a specified size. For each tile, the summary statistic (max, mean etc.) of the nearby values are output to a new feature map, and all other values are discarded.

- A common algorithm used for this process is called max pooling. Other pooling methods are **average pooling**, **weighted average pooling** etc.
- Pooling operation takes two parameters:
  - **size**: size of the tile to consider for pooling (typically 2×2 pixels)
  - **stride**: stride determines the locations where each tile is extracted.
- If we translate the input by a small amount, the values of most of the pooled outputs do not change, so pooling helps to make the model become more **invariant** to small translations of the input. We want our model to be invariant to many cases like:
  - **Location invariant :** Detect an edge/feature regardless of its location.
  - **Scale inavariant :** Detect an edge/feature regardless of its scale, but scale invariant cannot be acheived by pooling.
  - **Rotation invariant :** Detect an edge/feature regardless of its orientation.
- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. Popular pooling functions include :
  - **Max pooling :** reports the maximum output within a rectangular neighborhood. It can be thought of a kernel with stride and some padding. It introduce location invariance. In Max pooling we pick the maximum value among the values. We set a derivative value 1 to the max value and 0 to others. This is logical becasue the max value is not impacted by otherr values in the kernel.
  - **Average pooling :** of a rectangular neighborhood
  - **Weighted average pooling :** based on the distance from the central pixel (based on L_2 norm of a rectangular neighborhood)
- If we pool over the different convolved outputs separately, model can learn to be invariant to above transformations on the input.

- Pooling improves the computational efficiency of the network because the next layer has fewer inputs to process.

Fig. 16.5 shows how the pooling layer processes same input (number 5) with different location and orientation.

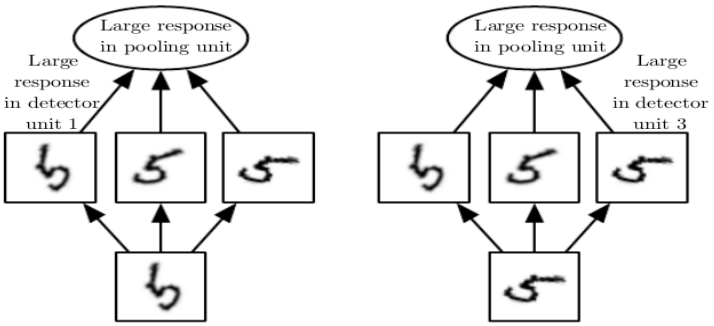


Figure 16.5: Example of learned invariances

16.4 Fully Connected Layers

At the end of a convolutional neural network are one or more **fully connected layers** (when two layers are “fully connected" every neuron in the first layer is connected to every neuron in the next layer). Their job is to perform classification or regression based on the features extracted by the convolutions layers.

- For classification models, typically, the final fully connected layer contains a softmax activation function.
- Before we feed the featues extracted by convolution layer to fully connected network, we need to do an operation on output of final convolution layer. This operation is called **flattening**.
  - In flattering, we convert a hiher order tensor into a vector. For eg: If we have max pooled output as a 3 order tensor, we can flatten it and the flattended layer is called **bottleneck layer** which is usually fed into fully connected layer.

16.5 Preventing Overfitting

As with any machine learning model, a key concern when training a convolutional neural network is overfitting. Two techniques to prevent overfitting when building a CNN are:

1. **Data augmentation**: Artificially boosting the diversity and number of training examples by performing random transformations to existing images to create a set of new variants. Data augmentation is especially useful when the original training data set is relatively small.

2. **Dropout regularization**: Randomly removing units from fully connected layers during a training gradient step.

Fig. 16.6 shows a data augmentation done on a cat image. he most popular tranformations done on image dataset are :

- Verticle / Horizontal shift
- Zoom in / Zoom out
- Rotation
- Flipping
- Add noise
- Shear : Strech one axis keeping the other same.

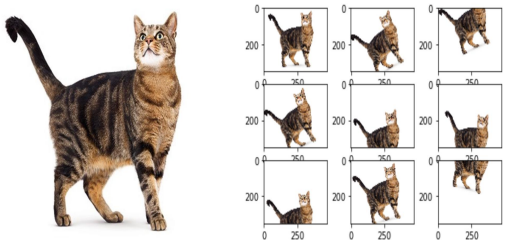


Figure 16.6: Data Augmentation

- Overfitting is more of a concern when working with smaller training data sets. When working with big data sets (e.g., millions of images), applying dropout is unnecessary, and the value of data augmentation is also diminished.

16.6 Leveraging Pretrained Models

Training a convolutional neural network from scratch, typically requires an extremely large amount of training data, and can be very time-consuming, taking days or even weeks to complete. But what if you could leverage existing image models trained on enormous datasets and adapt them for use in your own tasks?

- One common technique for leveraging pretrained models is feature extraction: retrieving intermediate representations produced by the pretrained model, and then feeding these representations into a new model as input.
- Usually the fully connected layer part is fine tuned to adapt the model to our task.

17

Recurrent Neural Network

Sequences they are everywhere. A speech consist of a sequence of words uttered, an email has sequence of words in it, a golf playing video captured has a sequence of images in it etc.

Figure 17.1: Sequence Data

The sequences has *many units which interact each other*, for example in speech data we have sequence of phonemes that relates to each other and gives us some meaningful content, video has image sequences where each image relates to the previous ones and gives us a meaningful related content. Most of the state of the art algorithm for speech recognition, language translation, video analysis are based on the sequence modelling task. The model that we have discussed so far is not capable of capturing the interactions of units. Sequence modelling ideas were developed to account for the unit interactions.

- In contrast to traditional machine learning models like logistic regression, MLP etc. in sequence modelling we assume the units we have at time step (t) depends on previous time steps (t-1, t-2, ...). So *Independent and Identically Distributed (IID)* assumption is no longer valid for sequence modelling. Here the order of the data points matter as order matters in a sequence.

The architecture of sequence modelling depends on the problem at hand. Below are the major architectures and real world applications where the sequence modelling is used. Fig. 17.2 three types of architectures.

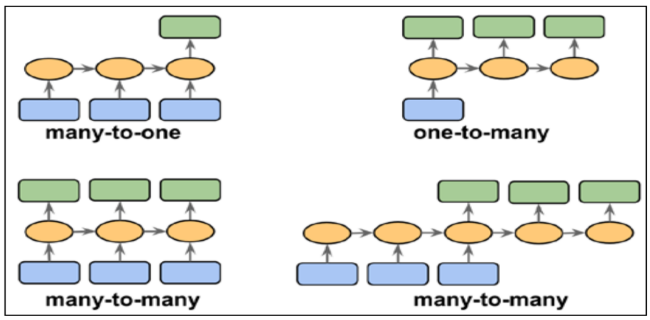


Figure 17.2: RNN Types

- **Many to One** : Input is a sequence of some lenght but output is of length one. For eg: Movie review, "This movie is too boring and long" - negative review. We have to get full input sequence before we make the prediction. Identifying language of an audio clip is another example.
- **One to Many** : Input is a single element and output is a sequence of elements. For eg: **Image captioning** where we feed a single image as input and output will be a sequence of words (sentence) describing the image.
- **Many to Many** : Here the input as well as output are sequences. There are two cases possible here:
  - **Same lenght sequences** : Output elements are generated immediatly after each input element is fed. eg: **Part of speech tagging** of a sentence. **Speech Recognition** is another applications where we need to identify the text corresponding to each word uttered.
  - **Different length sequences** : Ouput elements are generated after all the input elements are fed into the network. The input feeding stage is called **encoding** and output generation stage is called **decoding**. For eg: **Machine transalation** where we need to traslate from a language to another (eg: English to German)

17.1 Recurrent Neural Network

Suppose we are building an auto complete model where it fill in the next word you are going to type in. We don't know in advance how many words you are going to type in but wants to suggest the next word. Like many other sequence applications, here also the input size of the model is not fixed. Let's first try to solve this problem using MLP.

As MLP requires the input size to be fixed, lets set a vocabulary size for the words to be 100K, each word is represented by d dimesnions and the number of input neurons would be $100000d$, number of parameters to learn can become so huge too. This creates the burdern on training and also put the risk of overfitting. For better predicting the next word, we need to know what the current word is and the previous words too (i.e model has to memorize what happend previously) but MLP doesn't handle neither. How do we overcome this limitation? The solution is **Recurrent Neural Network (RNN)** which are neural network with memory.

Humans don't start their thinking from scratch every second. You don't throw everything away and start thinking from scratch again. Your thoughts have persistence and RNNs too.

In order to examine the architecture of RNNs, its memory and the flow of information in the network, a compact representation can be unfolded, which is shown in Fig. 17.3.

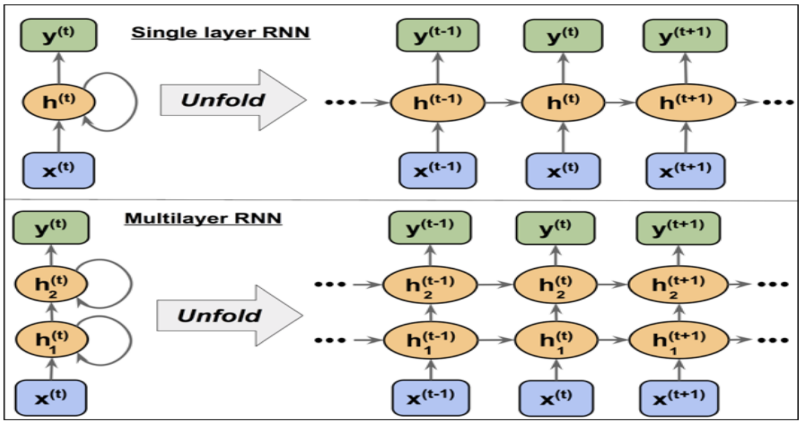


Figure 17.3: Unfolding RNN

- There is a feedback loop in the network (recurrent edge) that pasess the information of previous inputs (i.e memory). So this handles the need of memory mechanism in the model.
- In RNN, each unit has parameters which are shared across various time steps. This eliminates the problem of dealing with variable size input and also the overfitting due to huge number of parameters.
- In contrast to MLP hidden units, each hidden unit in RNN receives two distinct sets of inputs: 1-net input from downsteam layer, 2- activation of the same hidden unit but from the previous time step (state)

Now that we understand the structure and general flow of information in an RNN, let's get more specific and compute the actual activations of the

hidden layers as well as the output layer. For simplicity, we will consider just a single hidden layer; however, the same concept applies to multilayer RNNs.

Let $x^{(t)}, h^{(t)}, y^{(t)}$ denote the input, hidden and activation vectors at time step t and x, h, y be the dimensions of input word, number of hidden units and the dimension of output respectively. b_h, b_y are biases & $\phi_h(.), \phi_y(.)$ are activation functions of hidden and output layers respectively. Note that the weight, bias parameters between input, hidden, output layeres are shared across all timesteps. This is illusrated in Fig. 17.4

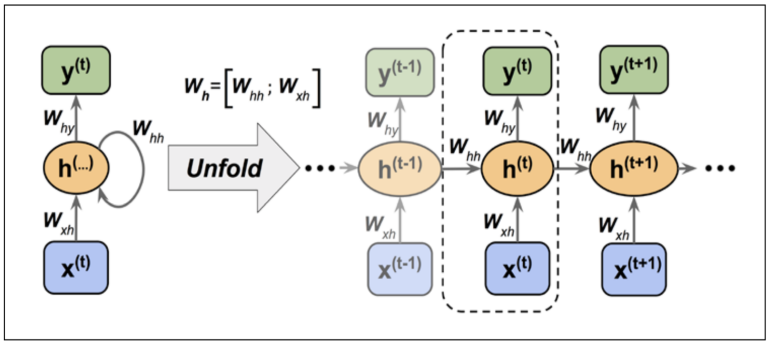


Figure 17.4: RNN Parameters

$$h^{(t)} = \phi_h\Big(W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h\Big) \tag{17.1}$$

$$y^{(t)} = \phi_y(W_{hy}h^{(t)} + b_y) \tag{17.2}$$

The learning algorithm for RNN is **Backpropagation Through Time** i.e backpropagation done at every time step. The derivation of the gradients is more tricky compared to normal backpropagation as the output of one time step depends on previous outputs and states. But the basic idea is that the overall loss L is the sum of all the loss values at times $t = 1$ to T.

$$L = \sum_{t=1}^{T} L^{(t)} \tag{17.3}$$

This long dependency put two challenges in training the RNN with backpropagation through time. We need to multiply weight parameters T times. For longer sequences T is large and depending on the magnitude of weights it can lead to **Vanishing Gradient** or **Exploding Gradient** as shown in Fig. 17.5.

Lets take an example of movie review "This movie is too boring in the first half but in the second half the hero did an amazing performance. I enjoyed it". The overall sentiment in the review is positive. If we go the

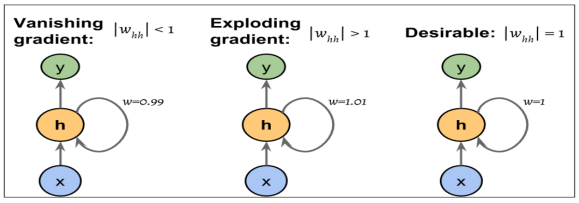


Figure 17.5: Vanishing Gradient & Exploding Gradient

review word by word this has negative sentiment words as well as positive sentiment words. We need some methods to **selectively read** positive words and **selectively forget** negative words. Also we want to pass the positive word state from previous step to the current step to memorize it. i.e **selective write**.

There are many NLP models based on this key idea. LSTMs, GRUs, Transformers, GPTs are sequence processing models based on this idea. These models have the mechanism to avoid / minimize the vanishing and exploding gradients.

18

Recommender Systems

How does YouTube know what video you might want to watch next? How does the Google play store pick an app just for you? - Its through recommender system. An ML-based recommendation model determines how similar videos and apps are to other things you like and then serves up a recommendation. Why recommender systems?

- A recommendation system helps users find compelling content in a large corpora. For example, the Google Play Store provides millions of apps, while YouTube provides billions of videos. More apps and videos are added every day. How can users find new and compelling content? Ofcourse one can search for it, but recommender systems finds the most relavant content automatically for you and shows up. The user may not be aware of all latest release of videos or apps but recommender systems makes it more convenient for the user to find it.
- Many brands/companies make huge revenue with recommender systems. Around 35% of e-commerce gaint Amazon's revenue is from product recommendation systems.

Two kinds of recommendations commonly used are:

- home page recommendations
  - *Recommended for you* These are personalized recommendations for a user based on their interest. Every user sees different recommendations
- related item recommendations
  - Thease are recommendations similar to a particular item, for eg: if you watched one video on YouTube, it shows up many simiar videos that you might be interested in.

Before we dive into the recommender systems, we should know a few important terminologies,

- **Item** or **Document**: The entities a system recommends. eg: Video, Apps etc.
- **Query** or **Context**: The information a system uses to make recommendations. Queries can be a combination of many information. eg: user id, items that user previously interacted with, user's device, time of day etc.
- **Embedding**: Converting Item and Query into vector representation. Many recommendation systems rely on learning an appropriate embedding representation of the queries and items.

18.1 Recommendation Systems Overview

One common architecture for recommendation systems consists of the following components:

- **Candidate Generation**:In this first stage, the system starts from a potentially huge corpus and generates a much smaller subset of candidates. eg: Reducing billions of YouTube videos to a few hundreds.
  - The model needs to evaluate queries quickly given the enormous size of the corpus. A given model may provide multiple candidate generators, each nominating a different subset of candidates.
- **Scoring**: This step scores and ranks the candidates in order to select the set of items (a few hundreds) to display to the user.
  - Since this model evaluates a relatively small subset of items, the system can use a more precise model relying on additional queries.
- **Re-ranking**: This step, take into account additional constraints for the final ranking. For example, the system removes items that the user explicitly disliked or boosts the score of fresher content.
  - Re-ranking can also help ensure diversity, freshness, and fairness.

18.2 Candidate Generation

Candidate generation is the first stage of recommendation. Given a query, the system generates a set of relevant candidates. The following table shows two common candidate generation approaches:

Type	Definition	Example
content-based filtering	Uses similarity between items to recommend items similar to what the user likes.	If user A watches two cute cat videos, then the system can recommend cute animal videos to that user.
collaborative filtering	Uses similarities between queries and items simultaneously to provide recommendations.	If user A is similar to user B, and user B likes video 1, then the system can recommend video 1 to user A (even if user A hasn't seen any videos similar to video 1).

18.2.1 Embedding Space

Both content-based and collaborative filtering map each item and each query to an embedding vector in a common embedding space $E = \mathbb{R}^d$. Typically, the embedding space is low-dimensional (that is, is much smaller than the size of the corpus), and captures some latent structure of the item or query set.

- In embedding method, similar items, such as YouTube videos that are usually watched by the same user, end up close together in the embedding space.
- How close is one item to the other is measured using **similarity measrues**.
- A similarity measure is a function $s : E \times E- > \mathbb{R}$ that takes a pair of embeddings and returns a scalar measuring their similarity.
- Embedding are used at candidate generation step, given a query embedding $q \in E$, the system looks for item embeddings $x \in E$ that are close to q, that is, embeddings with high similarity score $s(q, x)$.

To determine the degree of similarity $s(q, x)$ between vectors q, x, most recommendation systems rely on one or more of the following:

- **Cosine similarity**: $s(q, x) = cos(q, x)$. Items are ranked in the decreasing order of this score.
- **Dot product**: This score tells us how much two vectors point in the same direction. $s(q, x) = \sum_i = 1^d(q_i.x_i) = |q_i||x_i|cos(\theta)$, if the vectors are normalized then dot product is same as the cosine similarity. Items are ranked in decreasing order of this score.
  - Compared to the cosine, the dot product similarity is sensitive to the norm of the embedding. That is, the larger the norm of

an embedding, the higher the similarity (for items with an acute angle) and the more likely the item is to be recommended.

- Items that appear very frequently in the training set (for example, popular YouTube videos) tend to have embeddings with large norms. If capturing popularity information is desirable, then you should prefer dot product.
- One drawback with this metric is that, popular items may dominate due to high norm. In practice, you can use other variants of similarity measures that put less emphasis on the norm of the item. $|q_i|^\alpha |x_i|^\alpha cos(\theta)$, where $\alpha \in (0, 1)$.
- Items that appear very rarely may not be updated frequently during training. Consequently, if they are initialized with a large norm, the system may recommend rare items over more relevant items. To avoid this problem, be careful about embedding initialization, and use appropriate regularization.

- **Euclidian distance**: This distance represents the physical distance between vectors. When vectors are normalized this formula can be written in terms of dot product and cosine similarity. Items are ranked in increasing order of this score.

Fig. 18.1, shows the a query vector and three item vectors. To compare the three similarity metrics lets rank the item vectors based on three similarity scores.

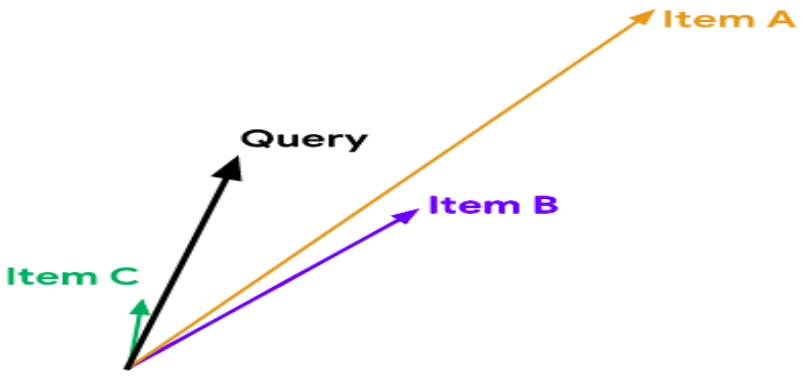


Figure 18.1: Similarity between query & item vectors

- Cosine similarity ranking : $C > A > B$ (angle increses in this order)
- Dot product ranking : $A > B > C$ (maginitude increases in this ordr)
- Euclidian ranking : $B > C > A$ (physical distances increases in this order)

18.2.2 Content-based Filtering

To domonstrate the content based recommender systems, lets take an example of google play store apps. Lets create a feature matrix for it, where the rows represent items (apps) and query (users). Columns represents the features like education, helath etc, user features could be a user selects "Entertainment apps" in their profile (explicit), user installed another app published by Science RUs (implicit - based on past action). For simplicity, lets assume the cell contents are binary - like user installed an app or not etc. Fig.18.2 illustrate this example.

Content-based filtering *uses item features* to recommend other items similar to what the user likes, based on their past actions or explicit feedback.

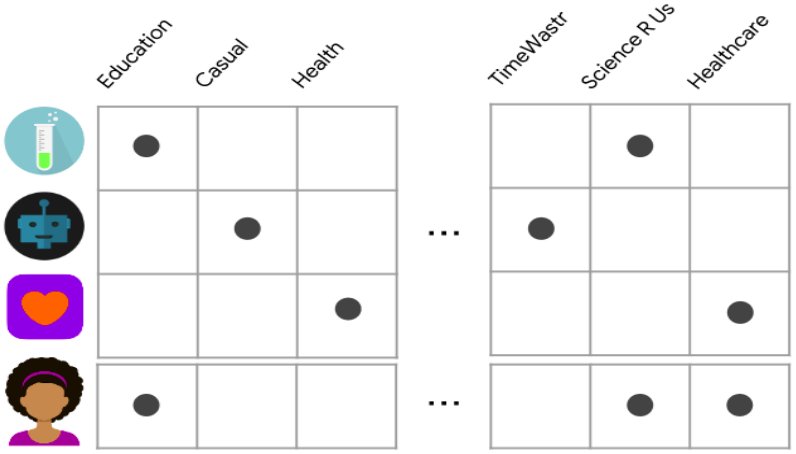


Figure 18.2: Content based system - Feature matrix

Note that the recommendations are specific to this user, as the model did not use any information about other users. The model should recommend items relevant to this user. To do so,

- We must first decide on a similarity metric (for example, dot product).
- As the next step, score each candidate item according to this similarity metric and rank it.
- Pick the top K items and recommend this to a user.

Content based similarity matrix comes with a few pros and cons. Table **??** has the details of pros and cons.

18.2.3 Collaborative Filtering

To address some of the limitations of content-based filtering, collaborative filtering uses similarities between users and items simultaneously to provide recommendations.

Advantages	Disadvantages
The model doesn't need any data about other users, since the recommendations are specific to this user. This makes it easier to scale to a large number of users.	Since the feature representation of the items are hand-engineered to some extent, this technique requires a lot of domain knowledge. Therefore, the model can only be as good as the hand-engineered features.
The model can capture the specific interests of a user, and can recommend niche items that very few other users are interested in.	The model can only make recommendations based on existing interests of the user. In other words, the model has limited ability to expand on the user's existing interests.

Table 18.1: Pros & Cons of Content based methods

- Collaborative filtering models can recommend an item to user A based on the interests of a similar user B.
- The embeddings can be learned automatically, without relying on hand-engineering of features.

Lets take a movie recommendation example to illustrate the idea of collaborative filtering. Here the training data is the feedback matrix, where rows represent the users nad columns represents the movie. The values in the training data can come from two things.

- **implicit**: This value the result of users implicit action. eg: User watched a movie because he liked it. So we can consider it as an implicit value
- **explicit**: This value is the result of users volunteer action. eg: User explicitly rated a movie

When a user visits the homepage, we should recommend movies based on movies that the user has watched previously and also based on similar users interests - i.e the movies liked by them.

- We represented both items and users in the same embedding space. This may seem surprising. After all, users and items are two different entities.
- However, we can think of the embedding space as an abstract representation common to both items and users, in which we can measure similarity or relevance using a similarity metric.

- The embeddings can be learned automatically, which is the power of collaborative filtering models. The collaborative nature of this approach is apparent when the model learns the embeddings.

- Embeddings of users with similar preferences will be close together and embeddings of movies liked by similar users will be close in the embedding space

18.3 Matrix Factorization

Matrix factorization is a simple embedding model. Given the feedback matrix $A \in \mathbb{R}^{N \times M}$, where N is the number of users (or queries) and M is the number of items, the model learns:

- A user embedding matrix $U \in \mathbb{R}^{N \times d}$, where row i is the embedding for user i.

- An item embedding matrix $V \in \mathbb{R}^{M \times d}$, where row j is the embedding for item j.

We can think a recommender system as rectangular matrix (feedback matrix) completion problem, where we have to fill in values for the cells with zero values (sparse cells). We can fill in the sparse cells by taking the product of factor matrices. i.e $\hat{A} = U.V^T$.

Fig. 18.3 shows the matrix factorization for movie feedback dataset. The embeddings are learned such that the product UV^T is a good approximation of the feedback matrix A. Observe that the (i, j) entry of $< U_i, V_j >$ is simply the dot product of the embeddings of user i and item j, which you want to be close to A_{ij}.

Figure 18.3: Matrix Factorization of Feature matrix

- Matrix factorization typically gives a more compact representation than learning the full matrix A with space $O(N \times M)$.
- Embedding space is only $O((M+N) \times d)$, where $d << M, N$ in most of real world dataset, as a result this factorization learns the latent structure.

To find the best value of d in matrix factorization we can use the knee method. We can do a plot by taking d on X-axis and error/ gradient on Y-axis. The d value for which the error is minimum or gradient is close to zero can be choosen as the optimum value of d.

18.3.1 Objective Function of Matrix Factorization

One intuitive objective function is the squared distance between the feedback matrix and the product of components U, V. There are three variants we can cosider for this optimization problem as shown in Fig. 18.4 and Table 18.2 compares three methods. Here *obs* represents the observed instances i.e we have a non-zero value in the feedback matrix.

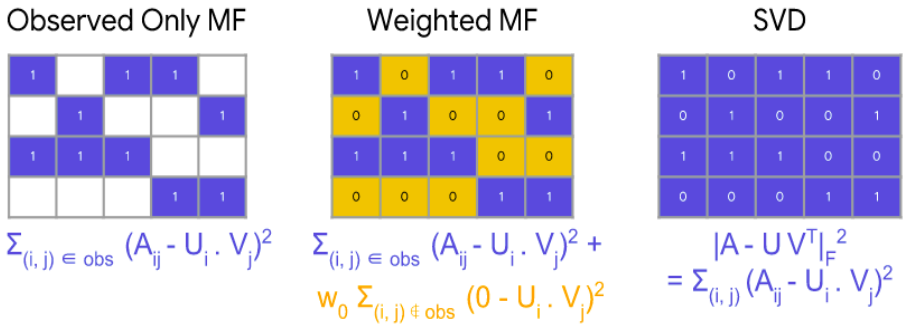


Figure 18.4: Matrix Factorization Methods

The weighted matrix factorization method is given in Eq. 18.1, where w_{ij} is function of user i and item j.

$$\min_{U \in \mathbb{R}^{N \times d},\ V \in \mathbb{R}^{M \times d}} \sum_{(i,j) \in \text{obs}} w_{ij}(A_{ij} - \langle U_i, V_j \rangle)^2 + w_0 \sum_{(i,j) \notin \text{obs}} (\langle U_i, V_j \rangle)^2 \quad (18.1)$$

The objective function is minimized using Stochastic Gradient Descend (SGD - Generic method) and **Weighted Alternate Least Squares** (WALS - specialized).

- WALS is prefered over SGD as it converger faster and easier to handle unobserved entires.

- WALS is reliant on square loss only - so its a specialized algorithm for matrix factorization minimization.

We have one heuristic way to address the cold start problem which is very common in recommender systems.

- **Averaging**: If the system does not have any interactions with a new item, then system can approximate its embedding by averaging the embeddings of items from the same category, from the same uploader (in YouTube), and so on.

18.4 Recommendation using Deep Neural Network

Deep Neural Network (DNN) based recommendations can address some of the limitations of Matrix Factorization method. DNNs can easily incorporate query features and item features (due to the flexibility of the input layer of the network), which can help capture the specific interests of a user and improve the relevance of recommendations. Some of the side featues that can be added with DNN model are age, country etc.

18.4.1 Softmax DNN for Recommendation

One possible DNN model is softmax, which treats the problem as a multi-class prediction problem in which:

- Input is the user query and the input features are of two types
  - densse features : eg: watch time & time since last watch
  - sparse features : eg: watch history & country
- Output is a probability vector with size equal to the number of items in the corpus, representing the probability to interact with each item; for example, the probability to click on or watch a YouTube video.

Fig. 18.5 shows the DNN architecture for recommender system, where $\psi(x) \in \mathbb{R}^d$ denote the output of the last layer and $V^{M \times d}$ is the matrix of weights of the softmax layer. A softmax function s maps this output to a probability distribution $\hat{p} = s(\psi(x)V^T)$. Let p be the ground truth probability distribution, ground truth could be YouTube videos that the user has clicked or watched.

The probability of an item j is given by $\hat{p}_j$,

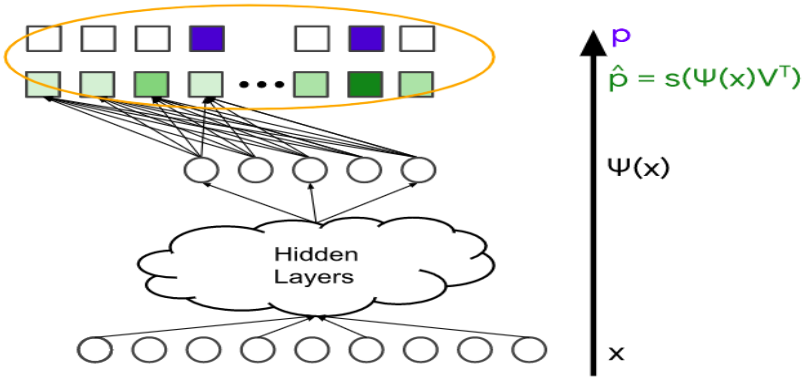


Figure 18.5: Recommender System - DNN model Architecture

$$\hat{p}_j = \frac{e^{\psi(x)V_j^T}}{Z}, \text{ where } Z \text{ is a constant} \tag{18.2}$$

$$Z = \sum_{i=1}^{M} e^{\psi(x)V_i^T} \tag{18.3}$$

$$\log \hat{p}_j = \psi(x)V_j^T - \log Z \tag{18.4}$$

- $\psi(x) \in \mathbb{R}^d$ is the embedding of query x
- $V_j^T \in \mathbb{R}^d$ is the vector of weights connecting the last hidden layer to output j. We call it the embedding of item j.
- So the *log probability of an item* is up to an additive constant from the dot product of item embedding and query embedding i.e $< \psi(x), V_j^T >$.
- Since log is an increasing function and highest value of $\hat{p}_j$ depends on the highest value of dot product $< \psi(x), V_j^T >$. So dot product can be interpretted as similarity measure in this embedding space.
- Since we are comparing two probability distributions p & $\hat{p}$ here, we can use the cross-entropy as a loss function for the model.

We have seen two methods for embeddings, one based on the matrix factorization in the previous section and one based on the DNN. How does these two methods compare,

- In both the softmax model and the matrix factorization model, the system learns one embedding vector V_j per item j (item embedding). What we called as the item embedding $V^{M \times d}$ is now the softmax layer weigts in DNN model.

- Instead of learning one embedding U_i per query i (query embedding) as in matrix factorization, DNN learns a mapping from the query feature x to an embedding $\psi(x) \in \mathbb{R}^d$.
  - We can think DNN model as a generalization of matrix factoriation, in which the query embedding is learned using a more generic non linear function $\psi(.)$.

Can we apply the idea of query embedding for the item embedding too? That is, instead of learning one embedding per item, can the model learn a nonlinear function that maps item features to an embedding? Yes we can !

- Here we need to use two neural network model, one for mapping query to a vector ($\psi(x_{query}) \in \mathbb{R}^d$) and another for mapping item to a vector ($\psi(x_{item}) \in \mathbb{R}^d$)
- Take the dot product of two mappings to get a score i.e $< \psi(x_{query}), \psi(x_{item}) >$. But this model is no longer a softmax model. This model output a scalar instead of a probability vector for a pair $(\text{query}, \text{item})$

When compared with matrix factorization methods, DNN based recommendation systems,

- can better capture personalized preferences, but are harder to train and more expensive to query (harder to scale to large corpora)
- can easily handle cold-start problem with new queries

18.5 Retrieval, Scoring and Re-Ranking

Suppose you have an embedding model (Matrix Factorization or DNN model). Given a user query, how would you decide which items to recommend? At serve time, given a query, you start by doing one of the following:

- For a matrix factorization model, the query (or user) embedding is known statically, and the system can simply look it up from the user embedding matrix.
- For a DNN model, the system computes the query embedding $q_i = \psi(x_i)$ for user features x_i at serve time by running the network on the feature vector x_i for user i.
- Search for an item embedding V_j that are close to q_i in the embedding space.
  - This is a k-nearest neighbor problem. We need to return top k similar items based on the similarity score between q_i, V_j

- We can use a similar approach in related-item recommendations. For example, when the user is watching a YouTube video, the system can first look up the embedding of that item, and then look for embeddings of other items V_j that are close in the embedding space.

After candidate generation, another model **scores** and ranks the generated candidates to select the set of items to display. The recommendation system may have multiple candidate generators that use different sources, such as the following:

- Related items from a matrix factorization model
- User features that account for personalization
- "Local" vs "distant" items; that is, taking geographic information into account
- Popular or trending items
- A social graph; that is, items liked or recommended by friends

The system combines these different sources into a common pool of candidates that are then scored by a single model and ranked according to that score. For example, the system can train a model to predict the probability of a user watching a video on YouTube given the following:

- query features (for example, user watch history, language, country, time)
- video features (for example, title, tags, video embedding)

The system can then rank the videos in the pool of candidates according to the prediction of the model. Since candidate generators compute a score (such as the similarity measure in the embedding space), you might be tempted to use them to do ranking as well. However, you should avoid this practice for the following reasons:

- Some systems rely on multiple candidate generators. The scores of these different generators might not be comparable.
- With a smaller pool of candidates, the system can afford to use more features and a more complex model that may better capture context.
- If the system always recommend items that are "closest" to the query embedding based only on one candidate generator, the candidates tend to be very similar to each other. This lack of diversity can cause a bad or boring user experience.

In the final stage of a recommendation system, the system can **re-rank** the candidates to consider additional criteria or constraints. One re-ranking approach is to use filters that remove some candidates.

- Training a separate model that detects whether a video is click-bait.
- system re-ranks videos by video age (perhaps to promote fresher content) & video length.
- model can re-rank candidates to incorporate freshness, diversity and fairness

Table 18.2: Comparison of Objecive functions

Observed only MF	**Weighted MF**	**SVD**
only summing over values of one is not a good idea. A matrix of all 1s will have a minimal loss and produce a model that can't make effective recommendations.	Weighted Matrix Factorization decomposes the objective into the following two sums: • A sum over observed entries. • A sum over unobserved entries (treated as zeroes).	We can treat the unobserved values as 0, and sum over all entries in the matrix A. This corresponds to minimizing the squared Frobenius distance between A and its approximation UV^T
	Frequent items (for example, extremely popular YouTube videos) or frequent queries (for example, heavy users) may dominate the objective function. • We can correct this problem by assigning different weights • These weights forms the hyperparameter and needs to be carefully tuned	we can solve this quadratic problem through Singular Value Decomposition (SVD) of the matrix. • SVD is not a great solution either, because in real applications, the matrix A may be very sparse. • Solution U, V will be will likely be close to zero, leading to poor generalization performance. • In contrast to PCA SVD decomposes a rectangular matrix.

Pros	**Cons**
We don't need domain knowledge because the embeddings are automatically learned. System needs only the feedback matrix and no contextual features are required.	The prediction of the model for a given (user, item) pair is the dot product of the corresponding embeddings. So, if an item is not seen during training, the system can't create an embedding for it and can't query the model (**cold start problem**) . In this situation, we use meta data such as location, gender, type of browser, type of device etc. the new user is using and from these meata information we can do a content based recommendation.
The model can help users discover new interests based on similar users interests.	**Side features** are any features beyond the query or item. Including available side features improves the quality of the model but its hard to inclde this type of features. For movie recommendations, the side features might include country or age.
	Popular items tend to be recommended for everyone, especially when using dot product as a similarity measure.It is better to capture specific user interests but matrix factorization doesn't have a way to capture side featuers

Table 18.3: Pros & Cons of Collaborative filtering methods