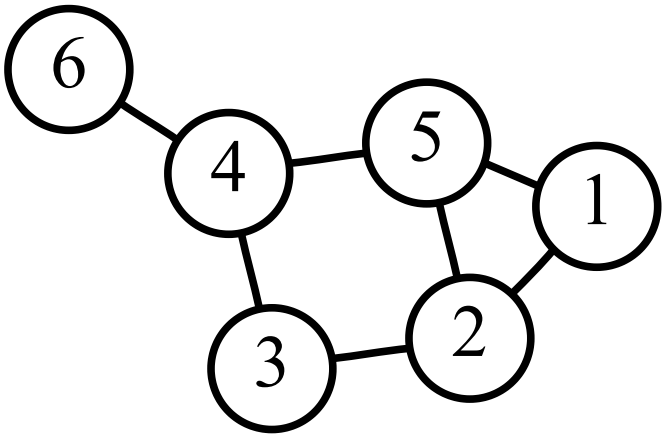
1. How can we represent a graph and what are its basic characteristics?

Brief Description:
A graph can be represented using adjacency matrices or adjacency lists. Basic characteristics include vertices, edges, degree, paths, and connections.

Detailed Description:
Graphs are structures used to model pairwise relations between objects. A graph G consists of vertices (nodes) V and edges (lines) E that connect pairs of vertices.



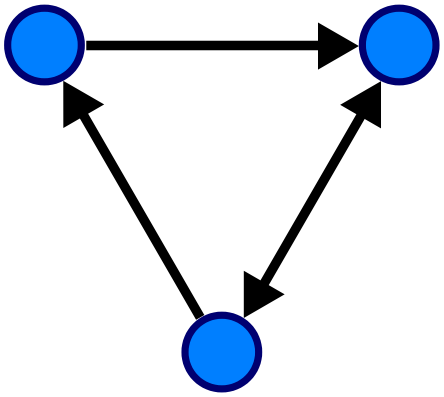
1. **Adjacency Matrix:** An $n \times n$ matrix where n is the number of vertices. The element $A[i][j]$ is 1 if there is an edge between vertex i and vertex j, otherwise it is 0.

$$A[i][j] = \begin{cases} 1 & \text{if there is an edge from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

2. **Adjacency List:** A collection of lists or arrays. The i-th list contains all vertices adjacent to the i-th vertex. This representation is space-efficient for sparse graphs.

Basic Characteristics:

- **Vertices (Nodes):** The fundamental units of a graph.
- **Edges (Links):** Connections between vertices.
- **Degree:** The number of edges connected to a vertex. For directed graphs, there are in-degrees and out-degrees.
- **Paths:** A sequence of vertices where each adjacent pair is connected by an edge.
- **Connectedness:** A graph is connected if there is a path between every pair of vertices.



A directed graph with three vertices and four directed edges (the double arrow represents an edge in each direction).

2. What is the Central Limit Theorem?

Brief Description:
The Central Limit Theorem (CLT) states that the distribution of the sum (or average) of a large number of independent, identically distributed random variables approaches a normal distribution, regardless of the original distribution of the variables.

Detailed Description:
The Central Limit Theorem is a fundamental theorem in probability theory and statistics. It provides the foundation for making inferences about population parameters based on sample statistics.

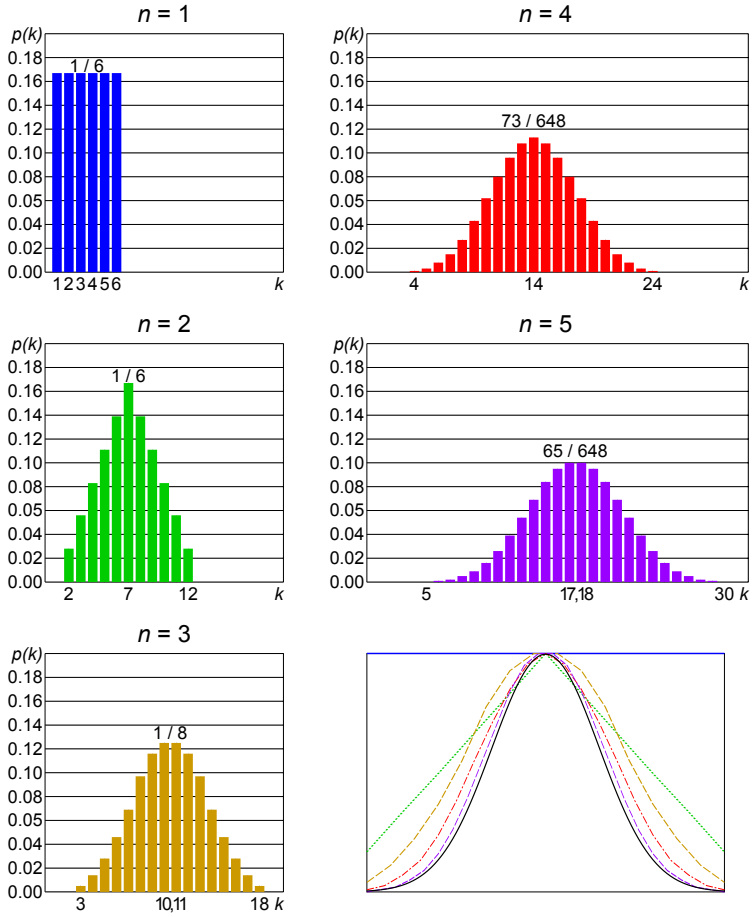
- **Formal Statement:**
  Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed $(i.i.d.)$ random variables with mean μ and variance σ^2 . As n approaches infinity, the sum (or average) of these variables:

  $$\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$$

  becomes approximately normally distributed with mean μ and variance $\frac{\sigma^2}{n}$:

  $$\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \xrightarrow{d} N(0,1)$$

  where $N(0,1)$ represents the standard normal distribution.
- **Implications:**
  The CLT allows us to use the normal distribution as an approximation for the distribution of the sample mean, even if the underlying distribution is not normal, provided the sample size is sufficiently large. This is particularly useful in hypothesis testing and constructing confidence intervals.



Comparison of probability density functions, p(k) for the sum of n fair 6-sided dice to show their convergence to a normal distribution with increasing n, in accordance to the central limit theorem. In the individual probability distribution functions, the minima, maxima and mods are labelled. In the bottom-right graph, smoothed profiles of the previous graphs are rescaled, superimposed and compared with a normal distribution, shown in black.

3. What is the Jaccard similarity?

Brief Description:

Jaccard similarity is a measure of similarity between two sets, defined as the size of the intersection divided by the size of the union of the sets. It ranges from 0 to 1, where 0 means no similarity and 1 means complete similarity.

Detailed Description:

The Jaccard similarity coefficient is used to compare the similarity and diversity of sample sets. It is formally defined as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

where:

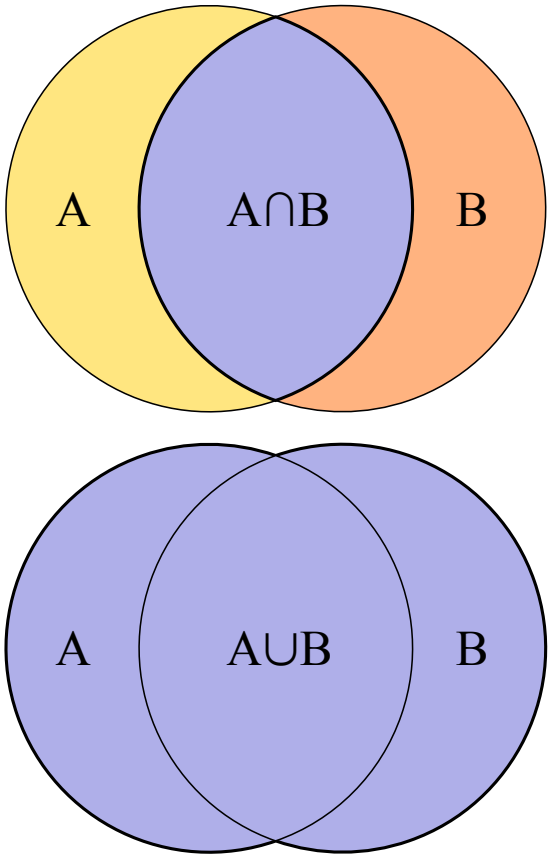
- A and B are two sets,
- $|A \cap B|$ is the cardinality of the intersection of sets A and B,
- $|A \cup B|$ is the cardinality of the union of sets A and B.

Properties:

- **Range:** The value of Jaccard similarity ranges between 0 and 1.
  - $J(A, B) = 0$ if A and B have no elements in common.
  - $J(A, B) = 1$ if A and B are identical.
- **Symmetry:** $J(A, B) = J(B, A)$.

Applications:

- **Document Similarity:** Comparing text documents to find duplicates or similar documents.
- **Recommender Systems:** Measuring similarity between users or items.
- **Clustering:** Grouping similar objects together based on their features. Jacard distance defined as:



Intersection and union of two sets A and B

4. What is a random variable? Give an example. What does it mean that two random variables are independent? What is the probability density function

(PDF) and what is the cumulative distribution function (CDF)? What is the relation between them?

Brief Description:

A random variable is a numerical outcome of a random phenomenon. Two random variables are independent if the occurrence of one does not affect the probability of occurrence of the other. The PDF describes the likelihood of a random variable taking on a particular value, while the CDF represents the cumulative probability that the variable will be less than or equal to a certain value.

Detailed Description:

- **Random Variable:**
  A random variable X is a variable whose value is subject to variations due to randomness. It can take on different values with certain probabilities.
  - **Example:** Consider a six-sided die roll. Let X be the random variable representing the outcome of the die roll. X can take on values {1, 2, 3, 4, 5, 6}.
- **Independence of Random Variables:**
  Two random variables X and Y are independent if the probability distribution of one variable is not affected by the value of the other. Formally, X and Y are independent if for all x and y:

$$P(X = x \text{ and } Y = y) = P(X = x) \cdot P(Y = y)$$

- **Probability Density Function (PDF):**
  The PDF of a continuous random variable X is a function $f_X(x)$ that describes the likelihood of X taking on a specific value x. The area under the PDF curve over an interval represents the probability that X falls within that interval.
  - **Equation:** For a continuous random variable X, the probability that X falls within the interval $[a, b]$ is given by:

$$P(a \leq X \leq b) = \int_a^b f_X(x)\,dx$$

- **Cumulative Distribution Function (CDF):**
  The CDF of a random variable X is a function $F_X(x)$ that represents the probability that X is less than or equal to a specific value x:

$$F_X(x) = P(X \leq x) = \int_{-\infty}^{x} f_X(t)\,dt$$

- **Relation Between PDF and CDF:**
  The PDF is the derivative of the CDF. Conversely, the CDF is the integral of the PDF. This relationship can be expressed as:

$$f_X(x) = \frac{d}{dx} F_X(x)$$

5. Formulate and explain the Bayes formula.

Brief Description:
Bayes' formula, or Bayes' theorem, provides a way to update the probability of a hypothesis based on new evidence. It expresses the conditional probability of an event based on prior knowledge of conditions related to the event.

Detailed Description:

- **Bayes' Theorem:**
  Bayes' theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event. It is mathematically expressed as:

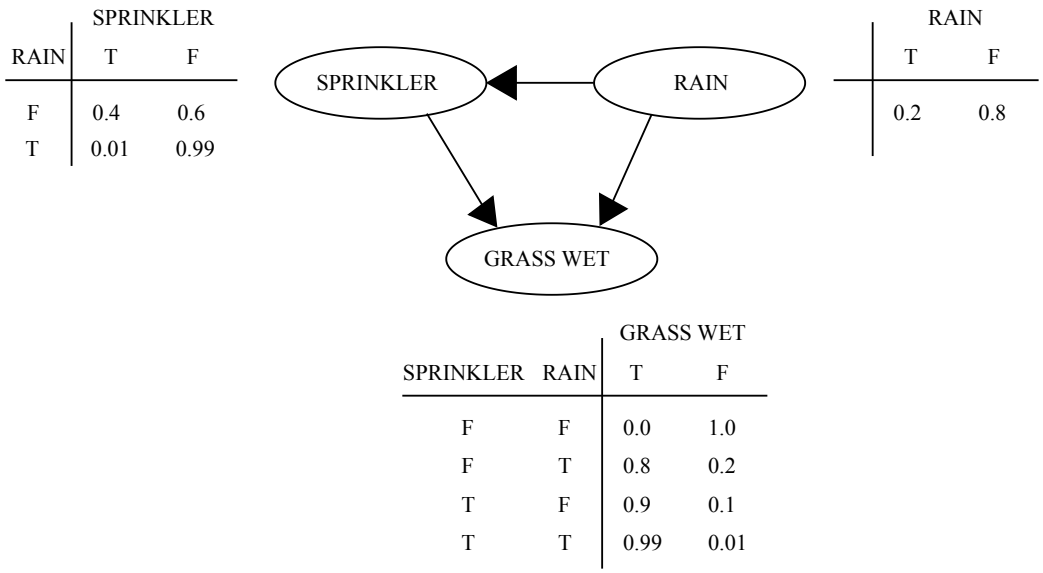
$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

  where:
  - $P(A|B)$ is the posterior probability of event A given that event B has occurred.
  - $P(A)$ is the prior probability of event A.
  - $P(B|A)$ is the likelihood of event B given that event A has occurred.
  - $P(B)$ is the marginal probability of event B.

Applications:

- **Medical Diagnosis:** Updating the probability of a disease given a positive test result.
- **Spam Filtering:** Determining the probability that an email is spam based on the presence of certain words.
- **Machine Learning:** Used in various algorithms, including Naive Bayes classifiers.

Example of Bayesian network:



A simple Bayesian network with conditional probability tables

6. Formulate Markov inequality.

Brief Description:
Markov's inequality provides an upper bound on the probability that a non-negative random variable exceeds a certain value. It is useful for deriving bounds in probability theory.

Detailed Description:

- **Markov's Inequality:**
  For a non-negative random variable X and any positive value a:

$$P(X \geq a) \leq \frac{\mathbb{E}[X]}{a}$$

  where $\mathbb{E}[X]$ is the expected value (mean) of X.

Markov's inequality (and other similar inequalities) relate probabilities to expectations, and provide bounds for the cumulative distribution function of a random variable. Markov's inequality can also be used to upper bound the expectation of a non-negative random variable in terms of its distribution function.

Implications:
Markov's inequality is a fundamental result in probability theory that provides a way to bound the tail probabilities of a distribution.

Example:
Let X be a random variable representing the amount of rainfall in a day, with an expected value of 10 mm. Using Markov's inequality, we can bound the probability that the rainfall exceeds 20 mm:

$$P(X \geq 20) \leq \frac{10}{20} = 0.5$$

7. Explain the basic properties of the PageRank algorithm.

Brief Description:
PageRank is an algorithm used by Google Search to rank web pages in their search engine results. It is based on the link structure of the web and assigns a numerical weighting to each element of a hyperlinked set of documents to measure its relative importance.

Detailed Description:

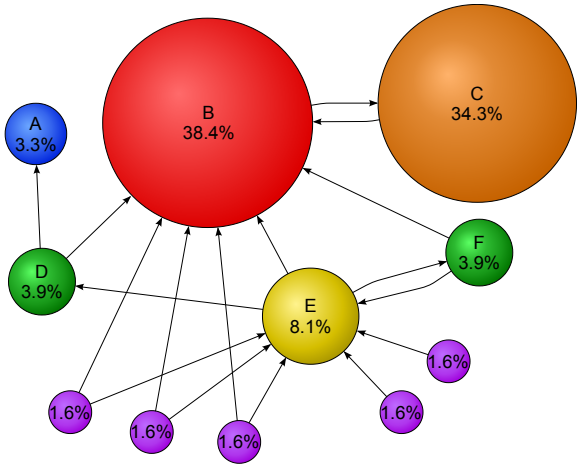
- **PageRank Algorithm:**
  PageRank works by counting the number and quality of links to a page to determine a rough estimate of the page's importance. The underlying assumption is that more important pages are likely to receive more links from other pages.
- **Properties:**
  - **Link Analysis:** Pages that are linked to by many pages or by important pages are assigned higher PageRank values.
  - **Damping Factor:** The algorithm includes a damping factor d (usually set to 0.85) which represents the probability that a user continues clicking on links. The remaining probability is used to model random jumps to any page.
  - **Recursive Calculation:** PageRank values are calculated iteratively until convergence.
- **Mathematical Formulation:**
  The PageRank of a page i is given by:

$$PR(i) = \frac{1-d}{N} + d \sum_{j \in M(i)} \frac{PR(j)}{L(j)}$$

  where:
  - N is the total number of pages,
  - $M(i)$ is the set of pages that link to page i,
  - $L(j)$ is the number of outbound links on page j,
  - d is the damping factor - probability that surfer will continue following links

Applications:

- **Search Engines:** Ranking web pages in search results.
- **Social Network Analysis:** Identifying influential users in a social network.
- **Recommendation Systems:** Recommending items based on their importance.



A simple illustration of the Pagerank algorithm. The percentage shows the perceived importance, and the arrows represent hyperlinks.

8. Explain K-means algorithm

Brief Description:

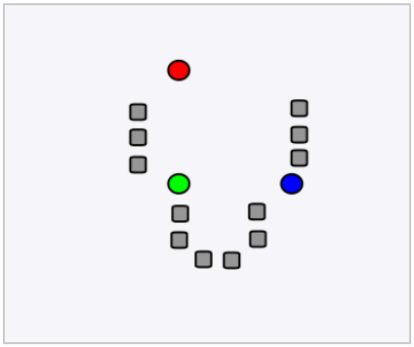
The K-means algorithm is a popular unsupervised learning method used for clustering data into K distinct non-overlapping subsets (clusters) based on the similarity of their features.

Detailed Description:

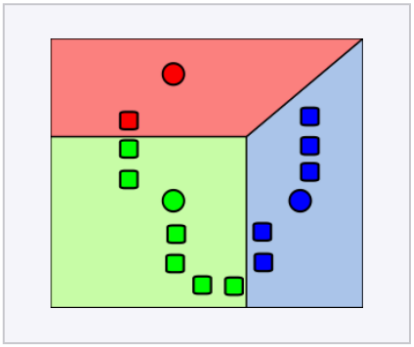
- **Objective:**
  The goal of K-means is to partition a dataset into K clusters in which each point belongs to the cluster with the nearest mean, serving as a prototype of the cluster.
- **Algorithm Steps:**
  i. **Initialization:** Select K initial centroids randomly from the dataset.
  ii. **Assignment Step:** Assign each data point to the nearest centroid based on the Euclidean distance. This forms K clusters.
  $\text{Assign } x_i \text{ to cluster } j \text{ if } \|x_i - \mu_j\|^2 \leq \|x_i - \mu_l\|^2 \text{ for all } l = 1, \ldots, K$
  where $\|x_i - \mu_j\|^2$ is the squared Euclidean distance between point x_i and centroid μ_j.
  iii. **Update Step:** Recalculate the centroids as the mean of all points assigned to each cluster.
  $\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$
  where C_j is the set of points in cluster j.
  iv. **Convergence Check:** Repeat the assignment and update steps until the centroids no longer change significantly (convergence) or a maximum number of iterations is reached.
- **Properties:**
  - **Efficiency:** K-means is computationally efficient and scales well with large datasets.

- **Simplicity:** The algorithm is straightforward to implement and understand.
- **Dependency on K:** The choice of K can significantly affect the results. Various methods like the Elbow Method can help determine the optimal number of clusters.
- **Sensitivity to Initial Centroids:** Different initializations can lead to different final clusters. Techniques like K-means++ can improve initialization.

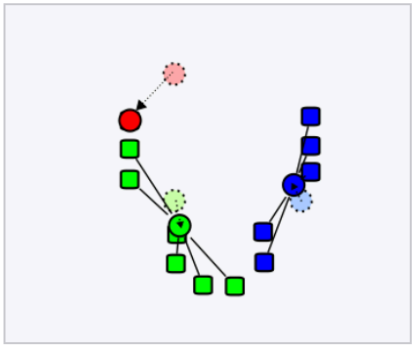
Demonstration of the standard algorithm



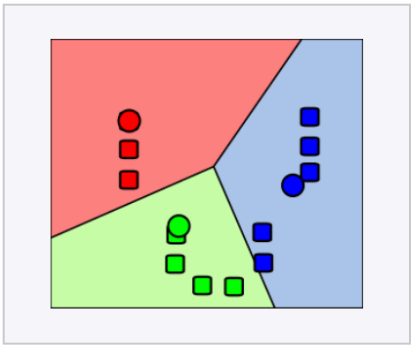
1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).



2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.



3. The centroid of each of the *k* clusters becomes the new mean.



4. Steps 2 and 3 are repeated until convergence has been reached.

9. What are the basic properties of a metric function?

Brief Description:
A metric function (distance function) defines a distance between elements of a set. It quantifies the "distance" between any two elements and must satisfy specific properties to be considered a valid metric.

Definition:
A metric on a set X is a function $d : X \times X \to \mathbb{R}$ that satisfies the following properties for all $x, y, z \in X$:

1. **Distance from a point to itself is 0:**

$$d(x, x) = 0$$

2. **Non-negative:**

$$d(x, y) \geq 0$$

   The distance between any two points is always positive.
3. **Symmetry:**

$$d(x, y) = d(y, x)$$

   The distance from x to y is the same as the distance from y to x.
4. **Triangle Inequality:**

$$d(x, z) \leq d(x, y) + d(y, z)$$

   The direct distance between two points is less than or equal to the sum of the distances when taking a detour through a third point.
   This is a natural property of both physical and metaphorical notions of distance: you can arrive at z from x by taking a detour through y, but this will not make your journey any shorter than the direct path.

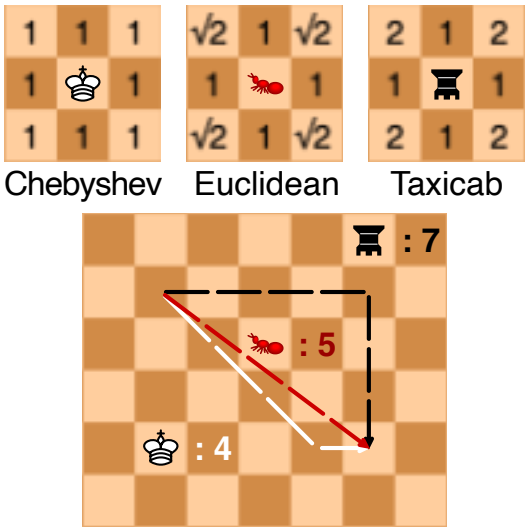
- **Examples:**
  - **Euclidean Distance:** In $\mathbb{R}^n$, the Euclidean distance between points $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ is:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

  - **Manhattan Distance (L1 Norm):** Also in $\mathbb{R}^n$, the Manhattan distance is:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

- **Hamming Distance:** For strings of equal length, the Hamming distance is the number of positions at which the corresponding symbols differ.
- **Edit distance**
- **Cosine similarity distance**
- **Jaccard distance**



Comparison of Chebyshev, Euclidean and taxicab distances for the hypotenuse of a 3-4-5 triangle on a chessboard.

10. Explain the basic assumptions of the MapReduce paradigm

Brief Description:
MapReduce is a programming model and processing technique for distributed computing based on parallel processing of data. It simplifies data processing on large clusters by breaking down tasks into a map and a reduce phase.

Detailed Description:

- **Basic Assumptions:**
  i. **Large-scale Data Processing:** The model is designed for processing large datasets that do not fit into a single machine's memory.
  ii. **Parallelism:** Data processing tasks can be divided into independent units that can be executed in parallel across multiple machines.
  iii. **Fault Tolerance:** The system is resilient to machine failures and can recover by re-executing failed tasks.
  iv. **Data Locality:** Processing tasks are moved to the location of the data to reduce network overhead and improve efficiency.

v. **Simple API:** The model provides a simple interface for users to define their data processing tasks using the map and reduce functions.

- **MapReduce Phases:**
  i. **Map Phase:** The input dataset is divided into smaller chunks, and the map function processes each chunk independently to produce intermediate key-value pairs.

$$\text{map}(k_1, v_1) \rightarrow [(k_2, v_2)]$$

  ii. **Shuffle and Sort Phase:** The intermediate key-value pairs are grouped by key, and the data is shuffled and sorted to prepare for the reduce phase.
  iii. **Reduce Phase:** The reduce function processes each group of key-value pairs to produce the final output.

$$\text{reduce}(k_2, [v_2]) \rightarrow [(k_3, v_3)]$$

- **Example: Word Count**
  - **Map Function:** Reads the input text and emits a key-value pair for each word.

$$\text{map}(\text{line}) \rightarrow (\text{word}, 1)$$

  - **Reduce Function:** Sums the counts for each word.

$$\text{reduce}(\text{word}, [1, 1, 1, \ldots]) \rightarrow (\text{word}, \text{total\_count})$$

- **Applications:**
  - **Data Processing:** Used for large-scale data processing tasks like log analysis, indexing, and data transformation.
  - **Machine Learning:** Supports distributed training and evaluation of machine learning models.
  - **Big Data Analytics:** Enables complex data analysis on massive datasets in distributed environments.

11. Name basic characteristics of stream algorithms. What algorithm uses the sliding window model?

Brief Description:

Stream algorithms are designed to process data streams where data arrives continuously and rapidly. They use limited memory and compute resources to provide approximate answers with high accuracy.

Detailed Description:

- **Basic Characteristics of Stream Algorithms:**

    i. **Single Pass:** Data is processed in a single pass or a few passes, as opposed to multiple passes over static datasets.
    ii. **Limited Memory:** Uses sublinear memory relative to the input size, often requiring only a fixed amount of memory.
    iii. **Real-time Processing:** Capable of processing data in real-time or near real-time.
    iv. **Approximation:** Provides approximate answers with provable error bounds due to memory and time constraints.
    v. **Scalability:** Scalable to large data volumes and high-velocity data streams.
- **Sliding Window Model:**
    - **Definition:** The sliding window model processes only the most recent data within a fixed-size window, discarding older data as new data arrives.
    - **Example Algorithm:** The Count-Min Sketch is a popular stream algorithm that can be adapted to use the sliding window model for frequency estimation of elements in the stream.
        - **Basic Idea:** The algorithm maintains a compact data structure to estimate the frequency of elements using hash functions. In the sliding window model, it adjusts the counts to ensure they only reflect the most recent data within the window.
    - **Another Algorithm:** Moving Average and other sliding algo like moving sum, max/min, median, frequency count

Example:
Consider monitoring the frequency of elements in a stream over the last n items. The sliding window model ensures that as new items arrive, the oldest items are removed from consideration.

- **Applications:**
    - **Network Monitoring:** Detecting anomalies and patterns in network traffic.
    - **Real-time Analytics:** Analyzing social media feeds, sensor data, and financial transactions.
    - **IoT:** Monitoring and analyzing data from IoT devices in real-time.

12. Give examples of procedural, object-oriented, and functional programming languages. Characterize them briefly.

Brief Description:
Programming languages can be categorized based on their programming paradigms, including procedural, object-oriented, and functional programming. Each paradigm offers a different approach to structuring and organizing code.

Detailed Description:

- **Procedural Programming Languages:**

    - **Examples:** C, Pascal, Fortran
    - **Characteristics:**
        - **Linear Structure:** Code is organized into procedures or functions, which are sequences of statements that perform specific tasks.
        - **Imperative Style:** Focuses on describing how a program operates, with explicit commands to change a program's state.
        - **Modularity:** Encourages the use of modular code by breaking down the program into smaller, reusable procedures.
        - **Global State:** Often relies on global variables and a shared state that can be accessed and modified by different parts of the program.
- **Object-Oriented Programming (OOP) Languages:**
    - **Examples:** Java, Python, C++
    - **Characteristics:**
        - **Encapsulation:** Combines data and functions that operate on the data into objects, encapsulating the internal state and behavior.
        - **Inheritance:** Allows the creation of new classes based on existing ones, promoting code reuse and hierarchical classification.
        - **Polymorphism:** Enables objects to be treated as instances of their parent class, allowing methods to be defined and used in multiple forms.
        - **Abstraction:** Hides complex implementation details and exposes only the necessary parts of an object.
- **Functional Programming Languages:**
    - **Examples:** Haskell, Lisp, Scala
    - **Characteristics:**
        - **Immutability:** Emphasizes the use of immutable data structures and avoids changing state or mutable data.
        - **First-Class Functions:** Functions are treated as first-class citizens, meaning they can be passed as arguments, returned from other functions, and assigned to variables.
        - **Higher-Order Functions:** Supports functions that operate on other functions, taking them as arguments or returning them as results.
        - **Pure Functions:** Functions that do not have side effects and always produce the same output for the same input, making reasoning about code easier.

13. What is i) eta-expansion, ii) lambda-function, and iii) currying? Give examples of these constructs in Scala.

Brief Description:

Eta-expansion, lambda-functions, and currying are functional programming constructs that are also used

in Scala. Each has its unique role in defining and manipulating functions.

Detailed Description:

- **i) Eta-Expansion:**
  - **Definition:** Eta-expansion is the process of converting a method into a function by adding an extra parameter list. It ensures that methods can be used where functions are expected.
  - **Example in Scala:**

```
def add(x: Int, y: Int): Int = x + y
val addFunction: (Int, Int) => Int = add _ // Eta-expansion
println(addFunction(3, 4)) // Output: 7
```

    In this example, the method `add` is eta-expanded to a function `addFunction`.
- **ii) Lambda-Function:**
  - **Definition:** A lambda-function (or anonymous function) is a function that is defined without being bound to an identifier. They are used to pass functions as arguments or return them as values.
  - **Example in Scala:**

```
val multiply = (x: Int, y: Int) => x * y // Lambda function
println(multiply(3, 4)) // Output: 12
```

    Here, `(x: Int, y: Int) => x * y` is a lambda function that multiplies two integers.
- **iii) Currying:**
  - **Definition:** Currying is the technique of transforming a function that takes multiple arguments into a sequence of functions, each with a single argument.
  - **Example in Scala:**

```
def add(x: Int)(y: Int): Int = x + y // Curried function
val addThree = add(3) _ // Partially applied function
println(addThree(4)) // Output: 7
```

    In this example, the function `add` is curried, allowing it to be partially applied.
    In mathematics and computer science, currying is the technique of translating a function that takes multiple arguments into a sequence of families of functions, each taking a single argument.

14. What problem solves the HyperLogLog algorithm? ii) What gave the name to the algorithm? iii) How the accuracy of the algorithm scales with the number of counters?

Brief Description:
The HyperLogLog algorithm is used to estimate the cardinality (number of distinct elements) of a large dataset efficiently. The name "HyperLogLog" is derived from its logarithmic space complexity properties.

Detailed Description:

- **i) Problem Solved by HyperLogLog Algorithm:**
  - **Cardinality Estimation:** HyperLogLog provides an efficient way to estimate the number of distinct elements in a dataset. It is particularly useful for large-scale data where exact counting is impractical due to memory constraints.
- **ii) Origin of the Name:**
  - **HyperLogLog:** The name reflects the algorithm's logarithmic properties. "Hyper" suggests the improvement over the previous LogLog algorithm, and "LogLog" indicates the double logarithmic space complexity.
- **iii) Accuracy and Scaling:**
  - **Accuracy:** The accuracy of the HyperLogLog algorithm improves with the number of counters (buckets). The standard error of the estimate is given by:

$$\text{Standard Error} \approx \frac{1.04}{\sqrt{m}}$$

    where m is the number of counters.
  - **Scaling:** Increasing the number of counters reduces the standard error, leading to more accurate cardinality estimates. However, this also increases the memory usage linearly.

Example:
Consider estimating the number of unique visitors to a website. Using HyperLogLog, you can maintain a compact data structure that provides an estimate with a controlled error margin, allowing you to handle large volumes of data efficiently.

15. What is entropy in physics and in information theory? Why is it important for data science?

Brief Description:
Entropy is a measure of disorder or randomness. In physics, it quantifies the amount of energy unavailable for doing work. In information theory, it measures the uncertainty or amount of information in a message.

Detailed Description:

- **Entropy in Physics:**
  - **Thermodynamic entropy:** Entropy is a thermodynamic quantity representing the amount of disorder or randomness in a system. It is denoted by S and is related to the number of possible microstates (W) of a system.

  The mathematical definition of thermodynamic entropy (S) for a reversible process is given by:

  $$dS = \delta Q/T$$

  where:
  δQ is the infinitesimal amount of heat added to the system reversibly.
  T is the absolute temperature of the system.
  For irreversible processes, the entropy increases, reflecting the natural tendency towards disorder.
  - **Statistical Physics entropy:**
    In statistical mechanics, entropy is related to the number of microscopic configurations (Ω) that correspond to a macroscopic state. Ludwig Boltzmann provided a statistical definition of entropy:

    $$S = k_B \ln \Omega$$

    where:
    - S is the entropy.
    - k_B is Boltzmann's constant.
    - Ω is the number of microstates consistent with the given macrostate.

  This definition bridges the microscopic and macroscopic descriptions of physical systems, explaining how macroscopic thermodynamic properties arise from microscopic behavior.
- **Entropy in Information Theory:**
  - **Definition:** In information theory, entropy measures the average amount of "information", "suprsie" or "uncertainty" produced by a stochastic source of data. It is denoted by H and quantifies the uncertainty or surprise associated with random variables.
  - **Equation:** The entropy H of a discrete random variable X with probability mass function $p(x)$ is:

    $$H(X) = -\sum_{x \in X} p(x) \log p(x)$$

  - **Importance:** Entropy helps in understanding data compression, transmission, and storage. It is fundamental in designing efficient coding schemes and in assessing the predictability of data.

  Example:
  Two bits of entropy: In the case of two fair coin tosses, the information entropy in bits is the base-2 logarithm of the number of possible outcomes—with two coins there are four possible outcomes, and two bits of entropy. Generally, information entropy is the average amount of information conveyed by an event, when considering all possible outcomes.

- **Importance for Data Science:**
  - **Decision Trees** connected with feature selection
  - **Feature Selection:** Entropy-based measures like information gain are used to select relevant features in machine learning models.
  - **Data Compression:** Entropy provides a theoretical limit on the best possible lossless compression of data.
  - **Uncertainty and Variability:** Understanding entropy helps in quantifying the uncertainty and variability in datasets, which is crucial for statistical modeling and hypothesis testing.

16. Models of Complex Networks and Their Typical Properties in Real Life

Brief Description:
Complex networks are systems with a large number of interconnected components, often modeled using graph theory. These models help to understand the structure and dynamics of various real-world systems, such as social networks, biological systems, and the internet.

Detailed Description:

1. **Introduction to Complex Systems:**
   - Complex systems consist of numerous interconnected parts that interact in non-trivial ways, leading to emergent behavior.
   - Examples include ecosystems, social networks, and the human brain.
2. **Power-Laws in Complex Systems:**
   - Many complex networks exhibit power-law distributions, where a few nodes have many connections (hubs), while most nodes have few.
   - This property is evident in networks such as the internet, where some websites (like Google) have a disproportionately high number of links.
3. **Cellular Automata:**
   - Cellular automata are discrete models used to simulate the behavior of complex systems.
   - Examples include Wolfram's one-dimensional system and the Game of Life, which demonstrate how simple rules can lead to complex behavior.
4. **Percolation and Criticality:**
   - Percolation theory studies the behavior of connected clusters in a random graph.
   - It provides insights into phenomena such as the spread of diseases or forest fires.
5. **Basic Measures and Theoretical Models:**
   - Empirical data on complex networks are analyzed using measures like degree distribution, clustering coefficients, and shortest path length.

- Theoretical models such as Erdős–Rényi (random graphs) and Barabási–Albert (scale-free networks) are used to describe network structures.

6. **Spreading Phenomena on Networks:**
   - Understanding how information, diseases, or behaviors spread through networks is crucial.
   - This includes studying viral marketing, epidemic modeling, and opinion dynamics.
7. **Agent-Based vs. Analytical Models:**
   - Agent-based models simulate the actions and interactions of autonomous agents to assess their effects on the system.
   - Analytical models use mathematical equations to describe the system's behavior.
   - Both approaches have advantages and disadvantages depending on the context.
8. **Modeling Tips:**
   - Successful modeling involves considering factors like initial conditions, averaging techniques, and updating schemes.
   - The choice between synchronous and asynchronous updating, as well as quenched vs. annealed approaches, can significantly impact model outcomes.
9. **Applications in Various Fields:**
   - In biology, complex networks can model interactions within cellular processes or ecosystems.
   - In social science, they can represent social interactions and communication patterns.
   - In economics, they help analyze market dynamics and financial networks.

Real-Life Properties of Complex Networks:

1. **Social Networks:**
   - Exhibit small-world properties, where most nodes can be reached from every other by a small number of steps.
   - High clustering coefficient and the presence of hubs are typical features.
2. **Biological Networks:**
   - Protein-protein interaction networks and metabolic networks show a hierarchical and modular structure.
   - These networks are robust to random failures but vulnerable to targeted attacks on highly connected nodes.
3. **Technological Networks:**
   - The internet and power grids are designed to be efficient yet robust.
   - These networks often have a scale-free structure with few highly connected hubs.
4. **Economic Networks:**
   - Represent the interactions between financial institutions, companies, and markets.
   - The interconnectedness can lead to cascading failures, highlighting the importance of network stability and resilience.

17. Classification of simple stationary points of a system of two autonomous first-order differential equations

Brief Description:

In the analysis of dynamical systems, stationary points (or equilibrium points) are where the system does not change, i.e., the derivatives are zero. For a system of two autonomous first-order differential equations, the classification of these points involves determining their stability and type (e.g., node, saddle, focus, center).

Detailed Description:

- **System of Two Autonomous First-Order Differential Equations:**
  Consider the system:

$$\begin{cases} \frac{dx}{dt} = f(x, y) \\ \frac{dy}{dt} = g(x, y) \end{cases}$$

  where f and g are functions of x and y.
- **Stationary Points:**
  Stationary points (x_0, y_0) satisfy:

$$f(x_0, y_0) = 0 \quad \text{and} \quad g(x_0, y_0) = 0$$

- **Linearization:**
  Near a stationary point, the system can be linearized using the Jacobian matrix J:

$$J = \left(\begin{matrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{matrix} \right) \Bigg|_{(x_0, y_0)}$$

- **Eigenvalues of the Jacobian:**
  The behavior of the system near a stationary point is determined by the eigenvalues λ_1 and λ_2 of J.
- **Classification:**
  - **Node:**
    - **Stable Node:** $\lambda_1 < 0$ and $\lambda_2 < 0$
      Trajectories converge to the stationary point as time goes to infinity. The system returns to equilibrium after small perturbations.
    - **Unstable Node:** $\lambda_1 > 0$ and $\lambda_2 > 0$
      Trajectories diverge from the stationary point as time goes to infinity. The system moves away from equilibrium after small perturbations.
  - **Saddle Point:** λ_1 and λ_2 have opposite signs ($\lambda_1 > 0, \lambda_2 < 0$ or $\lambda_1 < 0, \lambda_2 > 0$)
    Trajectories are attracted to the point along one axis and repelled along another. The system

exhibits both stable and unstable behavior depending on the direction of the perturbation.

- **Focus (Spiral Point):**
  - **Stable Focus:** Complex eigenvalues with negative real parts ($\lambda = \alpha \pm i\beta$ with $\alpha < 0$) Trajectories spiral into the stationary point as time goes to infinity. The system returns to equilibrium with a spiraling motion.
  - **Unstable Focus:** Complex eigenvalues with positive real parts ($\lambda = \alpha \pm i\beta$ with $\alpha > 0$) Trajectories spiral out of the stationary point as time goes to infinity. The system moves away from equilibrium with a spiraling motion.
- **Center:** Purely imaginary eigenvalues ($\lambda = \pm i\beta$).
  The trajectories are closed orbits (ellipses or circles). Trajectories form closed orbits (ellipses or circles) around the stationary point. The system exhibits periodic behavior and remains at a constant distance from the equilibrium.

Example 1:
Consider the system:

$$\begin{cases} \frac{dx}{dt} = x - y^2 \\ \frac{dy}{dt} = x + y^2 \end{cases}$$

The Jacobian matrix at the stationary point (0, 0) is:

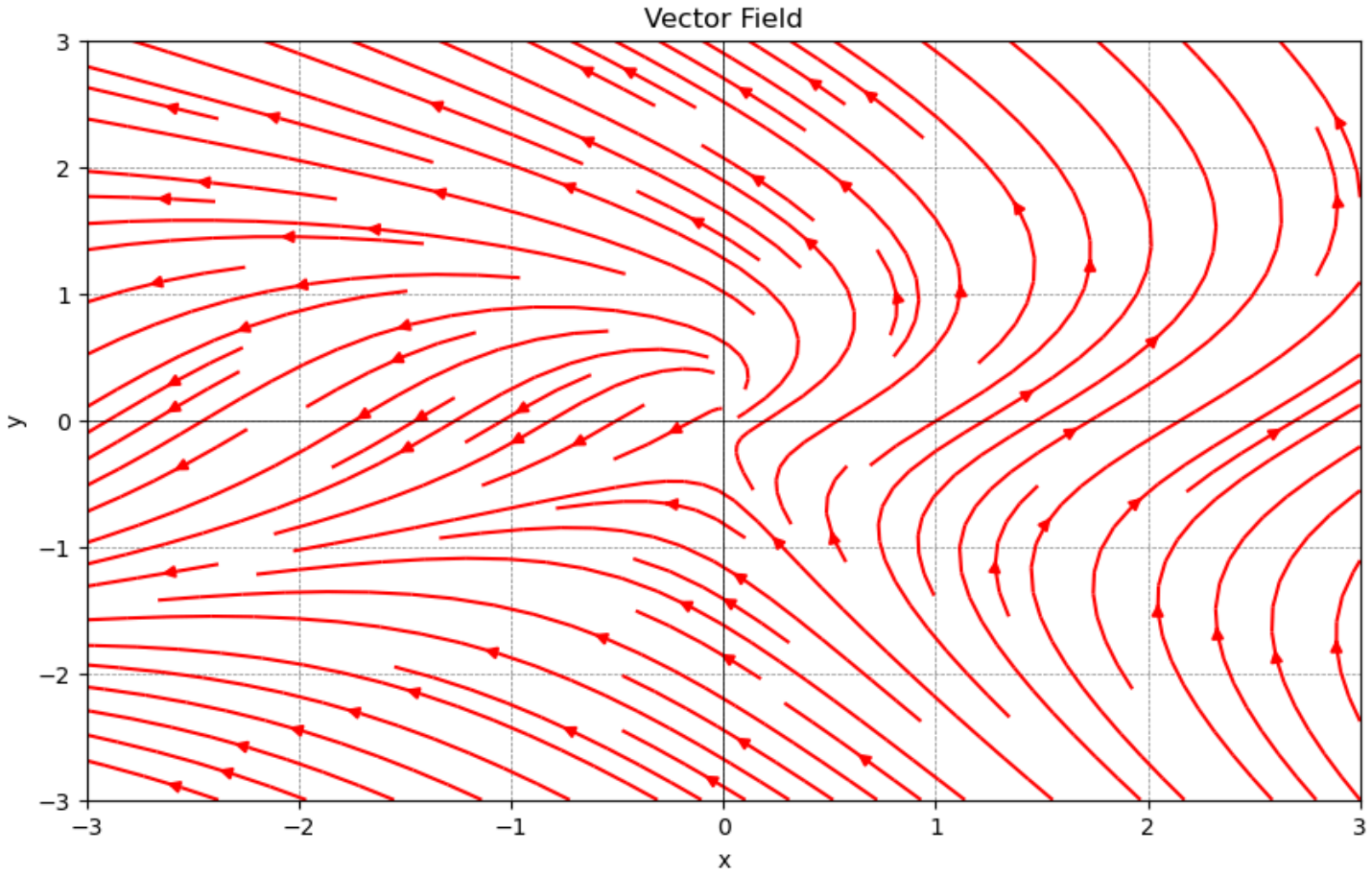
$$J = \begin{pmatrix} \frac{\partial(x-y^2)}{\partial x} & \frac{\partial(x-y^2)}{\partial y} \\ \frac{\partial(x+y^2)}{\partial x} & \frac{\partial(x+y^2)}{\partial y} \end{pmatrix} = \begin{pmatrix} 1 & -2y \\ 1 & 2y \end{pmatrix}$$

At the stationary point (0, 0):

$$J = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

The eigenvalues of this matrix are $\lambda = 1$ and $\lambda = 0$. The presence of a zero eigenvalue suggests that further analysis is needed, but generally, this indicates a line of stationary points rather than an isolated point. This may require considering higher-order terms or a different analytical approach to fully classify the stability.

Stationary Points: [(0, 0)]
Classification: Saddle Point



Example 2:

Consider the system:

$$\begin{cases} \frac{dx}{dt} = y + x(1 - x^2 - y^2) \\ \frac{dy}{dt} = -x + y(1 - x^2 - y^2) \end{cases}$$

The Jacobian matrix at a stationary point (x_0, y_0) is:

$$J = \begin{pmatrix} \frac{\partial(y+x(1-x^2-y^2))}{\partial x} & \frac{\partial(y+x(1-x^2-y^2))}{\partial y} \\ \frac{\partial(-x+y(1-x^2-y^2))}{\partial x} & \frac{\partial(-x+y(1-x^2-y^2))}{\partial y} \end{pmatrix}$$

Stationary Points: [(0, 0)]

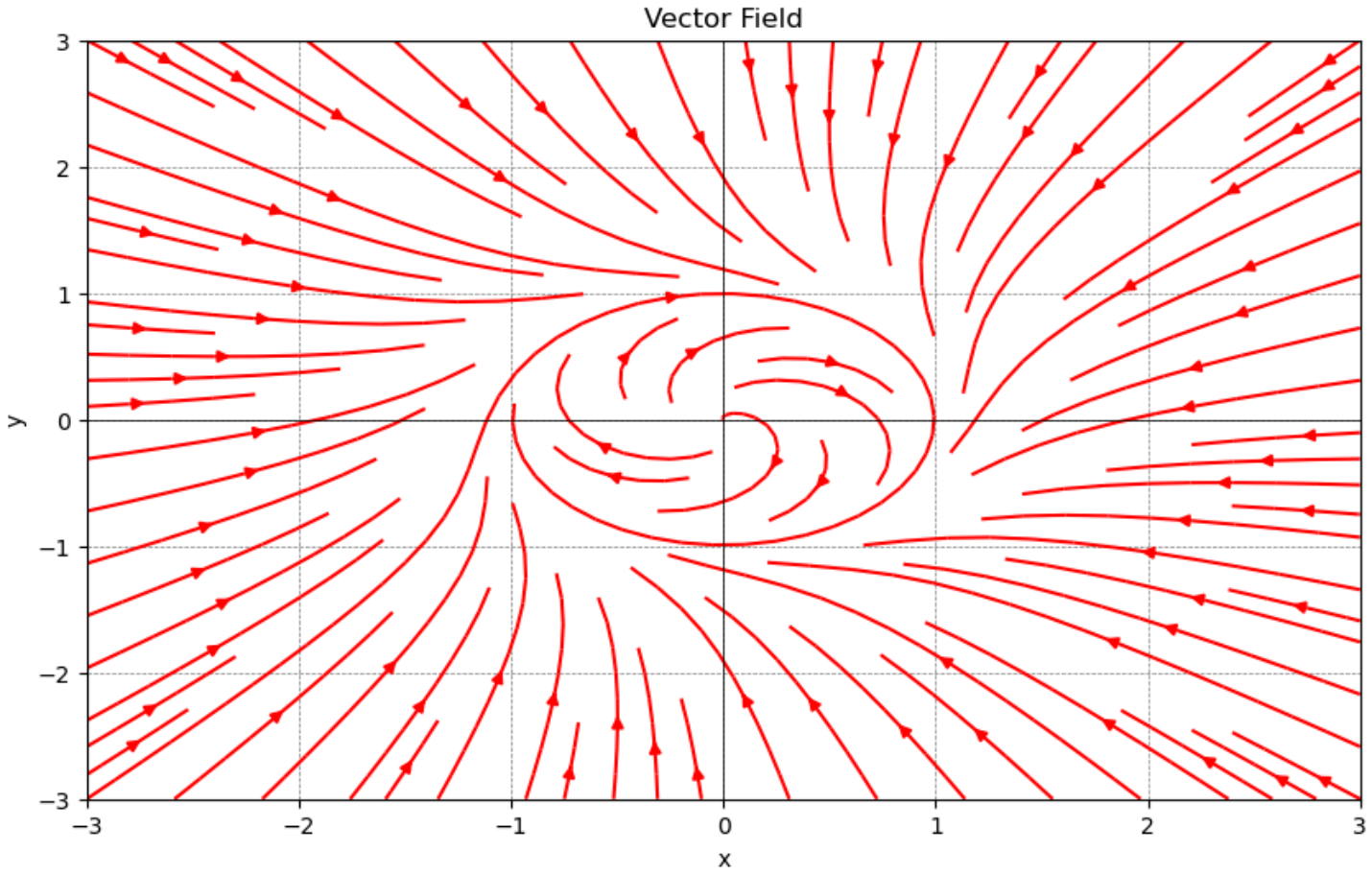
Jacobian Matrix:

$$J = \begin{pmatrix} 1 - 3x^2 - y^2 & 1 - 2xy \\ -1 - 2xy & 1 - x^2 - 3y^2 \end{pmatrix}$$

Stationary Point: (0, 0)

Jacobian Matrix at (0, 0):

Matrix(1, 1], [-1, 1)
Eigenvalues: {1 - I: 1, 1 + I: 1}
Classification: Unstable Focus (Spiral)



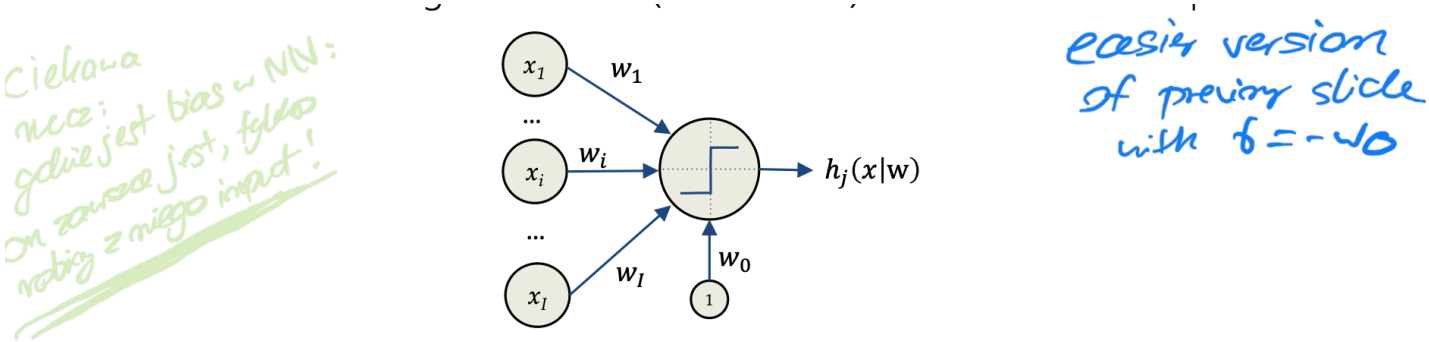
18. Explain the perceptron algorithm. What is overfitting and how to prevent it?

Brief Description:
The perceptron algorithm is a supervised learning algorithm used for binary classification. It aims to find a hyperplane that separates data into two classes. Overfitting occurs when a model learns the noise in the training data rather than the underlying pattern. Various techniques can help prevent overfitting.

Detailed Description:

- **Perceptron Algorithm:**
  - **Definition:** The perceptron is a type of linear classifier, an algorithm for supervised learning of binary classifiers.
  - **Model:** The perceptron model represents a binary classifier using a linear decision boundary.
    $$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$
    where w is the weight vector, x is the input vector, and b is the bias term.

- **Algorithm Steps:**
  a. **Initialization:** Initialize the weights w and bias b to small random values or zeros (initialization for boundary conditions is not always a good idea!). Bias can be treated as w0 weight which is also updated!
  b. **Prediction:** For each training example x_i:
  $$y_i = \begin{cases} 1 & \text{if } w \cdot x_i + b > 0 \\ 0 & \text{otherwise} \end{cases}$$
  c. **Update Rule:** Update the weights and bias based on the prediction error:
  $w = w + \eta(y_i - \hat{y}_i)x_i$
  where η is the learning rate, y_i is the actual label, and $\hat{y}_i$ is the predicted label.
  d. **Iteration:** Repeat the prediction and update steps for a fixed number of iterations or until convergence.



$$h_j(x|\mathrm{w},\mathrm{b}) = \mathrm{h_j}\left(\Sigma_{i=1}^{I} w_i \cdot x_i - b\right) = \mathrm{h_j}\left(\Sigma_{i=0}^{I} w_i \cdot x_i\right) = h_j(w^T x)$$

- **Overfitting:**
  - **Definition:** Overfitting occurs when a model learns the training data too well, capturing noise and outliers instead of the underlying pattern. This results in poor generalization to new, unseen data.
  - **Symptoms:**
    - High accuracy on training data but poor performance on validation/test data.
    - The model is overly complex with many parameters relative to the number of training samples.
- **Preventing Overfitting:**
  i. **Cross-Validation:**
    - Use techniques like k-fold cross-validation to ensure the model generalizes well to unseen data.
  ii. **Regularization:**
    - Add a penalty term to the loss function to discourage overly complex models. Common methods include L1 (Lasso) and L2 (Ridge) regularization.

$$\text{L2 Regularization: } J(w) = \sum_i (y_i - \hat{y}_i)^2 + \lambda \|w\|^2$$

iii. **Pruning:**
   - Simplify the model by removing less important features or parameters, especially in decision trees and neural networks.

iv. **Early Stopping:**
   - Monitor the model's performance on a validation set during training and stop when performance starts to degrade.

v. **Data Augmentation:**
   - Increase the size and diversity of the training data by applying transformations such as rotations, scaling, and flips (especially in image data).

vi. **Dropout:**
   - In neural networks, randomly drop units (along with their connections) during training to prevent co-adaptation.

19. How do Support Vector Machines work?

Brief Description:
Support Vector Machines (SVMs) are supervised learning models used for classification and regression tasks. SVMs aim to find the optimal hyperplane that maximizes the margin between different classes in the feature space.

Detailed Description:

- **Objective:**
  The primary objective of SVMs is to find a hyperplane that separates data points of different classes with the largest possible margin. The hyperplane is chosen such that the distance (margin) between the hyperplane and the nearest data points from each class (support vectors) is maximized.
- **Linear SVM:**
  For linearly separable data, the SVM algorithm finds a hyperplane defined by:

$$w \cdot x + b = 0$$

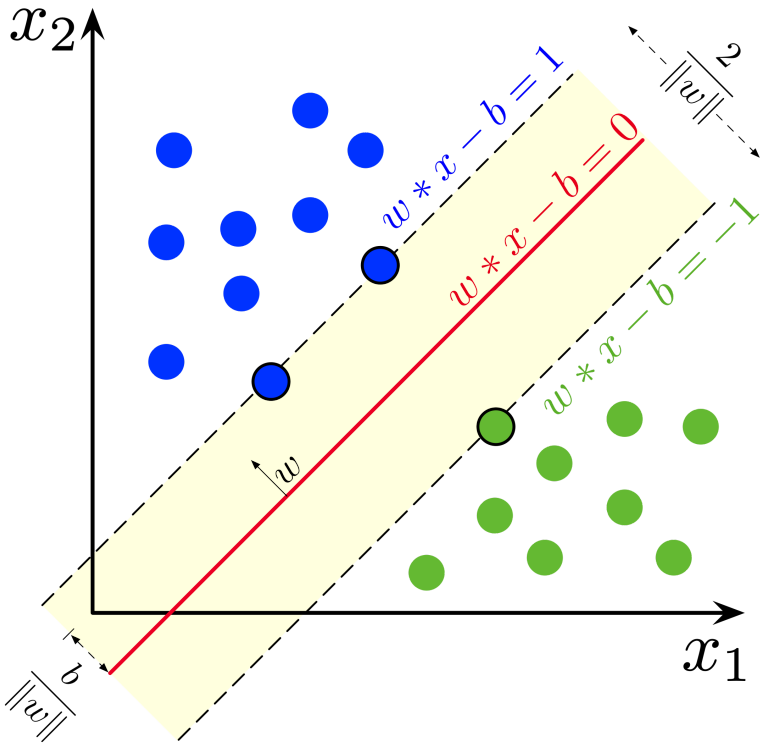
  where w is the weight vector, x is the feature vector, and b is the bias term.
  The goal is to maximize the margin $\frac{2}{\|w\|}$, subject to the constraints:

$$y_i(w \cdot x_i + b) \geq 1 \quad \forall i$$

  where y_i are the class labels ($+1$ or -1) and x_i are the data points.
  The optimization problem can be formulated as:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{subject to} \quad y_i(w \cdot x_i + b) \geq 1$$

Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors:

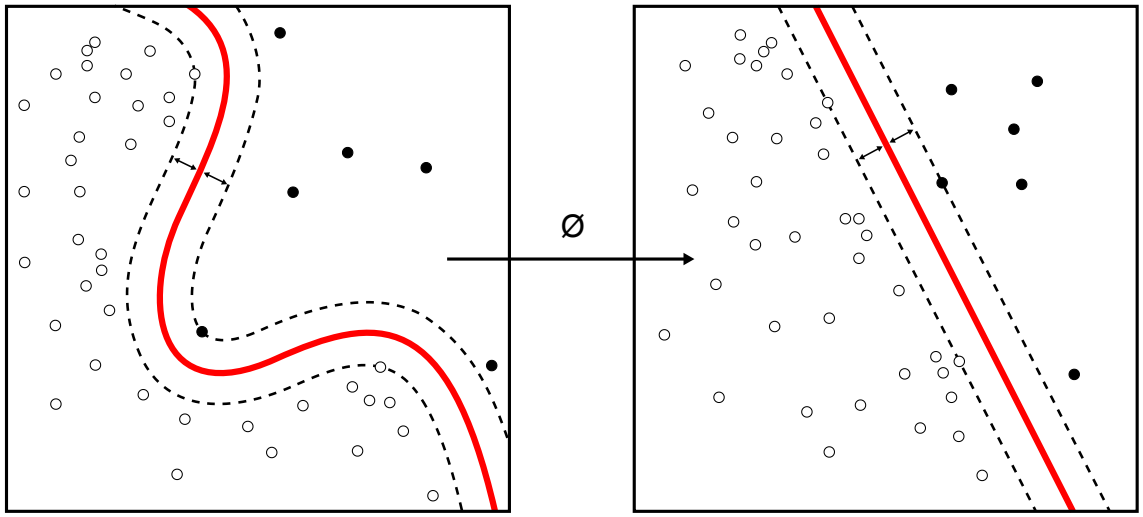


- **Non-linear SVM:**
  For non-linearly separable data, SVMs use kernel functions to map the input features into a higher-dimensional space where a linear hyperplane can separate the classes. Common kernel functions include:
  - **Polynomial Kernel:** $K(x_i, x_j) = (x_i \cdot x_j + 1)^d$
  - **Radial Basis Function (RBF) Kernel:** $K(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2)$
  - **Sigmoid Kernel:** $K(x_i, x_j) = \tanh(\kappa x_i \cdot x_j + \theta)$

  The optimization problem becomes:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to} \quad y_i(w \cdot \phi(x_i) + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0$$

  where ξ_i are slack variables allowing some misclassification, C is the regularization parameter, and $\phi(x_i)$ is the feature transformation function defined by the kernel.



kernel machine

20. Discuss dimensionality reduction techniques.

Brief Description:

Dimensionality reduction techniques are used to reduce the number of features in a dataset while preserving as much information as possible. These techniques help to simplify models, reduce computational cost, and mitigate the curse of dimensionality.

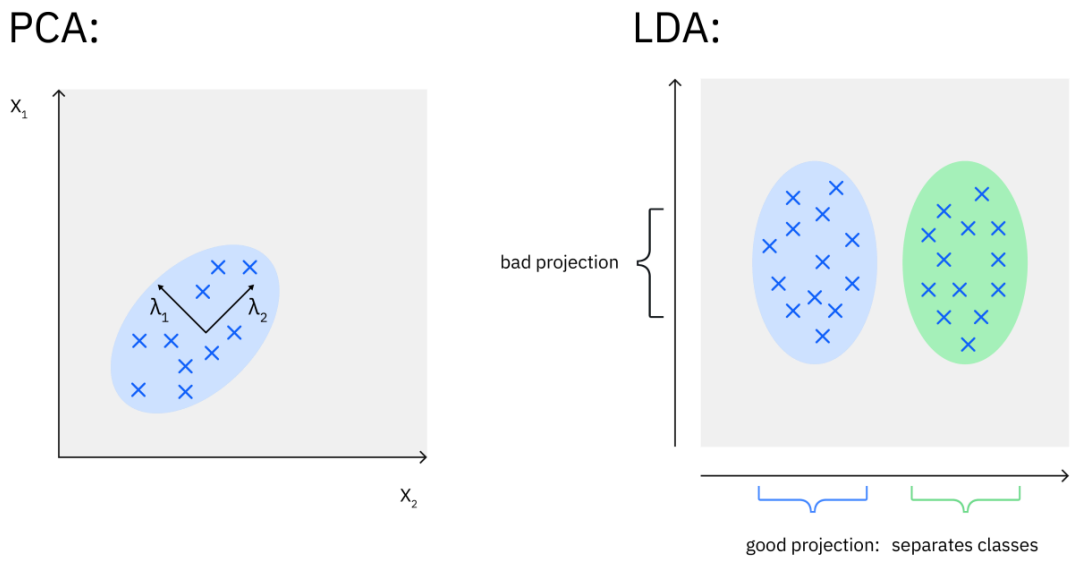
Detailed Description:

- **Feature Selection:**
  Feature selection approaches try to find a subset of the input variables (also called features or attributes). The three strategies are: the filter strategy (e.g. information gain), the wrapper strategy (e.g. search guided by accuracy), and the embedded strategy (selected features are added or removed while building the model based on prediction errors).
- **Principal Component Analysis (PCA):**
  - **Definition:** PCA is a linear technique that transforms the original features into a new set of orthogonal features (principal components) that capture the maximum variance in the data.
  - **Steps:**
    a. Standardize (normalize) the data.
    b. Compute the covariance matrix.
    c. Calculate the eigenvalues and eigenvectors of the covariance matrix.
    d. Sort the eigenvalues and select the top k eigenvectors.
    e. Transform the original data to the new subspace.
- **Linear Discriminant Analysis (LDA):**
  - **Definition:** LDA is a supervised technique that aims to maximize the separation between multiple classes. It projects the data onto a lower-dimensional space that maximizes the ratio of between-

class variance to within-class variance.

- **Steps:**
  a. Compute the within-class and between-class scatter matrices.
  b. Calculate the eigenvalues and eigenvectors of the scatter matrices.
  c. Select the top k eigenvectors to form a transformation matrix.
  d. Transform the original data to the new subspace.

Linear discriminant analysis (LDA) is similar to PCA in that it projects data onto a new, lower dimensional space, the dimensions for which are derived from the initial model. LDA differs from PCA in its concern for retaining classification labels in the dataset. While PCA produces new component variables meant to maximize data variance, LDA produces component variables that also maximize class difference in the data. Steps for implementing LDA are similar to those for PCA. The chief exception is that the former uses the scatter matrix whereas the latter uses the covariance matrix. Otherwise, much as in PCA, LDA computers linear combinations of the data's original features that correspond to the largest eigenvalues from the scatter matrix. One goal of LDA is to maximize interclass difference while minimizing intraclass difference.



- **t-Distributed Stochastic Neighbor Embedding (t-SNE):**
  - **Definition:** t-SNE is a non-linear technique primarily used for visualizing high-dimensional data. It minimizes the divergence between two distributions: one representing pairwise similarities in the original space and the other in the reduced space. It works like a physical system on spings which is evolving during the time to some optimal points.
- **Autoencoders:**
  - **Definition:** Autoencoders are neural networks used for non-linear dimensionality reduction. They consist of an encoder that maps the input to a lower-dimensional latent space and a decoder that reconstructs the input from the latent representation.

- **Radnom Forest:** Random forests provide an inherent feature importance ranking, which helps in identifying the most relevant features. This is achieved through the following methods:
  Mean Decrease Impurity (MDI): Also known as Gini importance, this measures the average decrease in impurity (e.g., Gini impurity for classification tasks) each feature contributes across all trees in the forest.
  Mean Decrease Accuracy (MDA): This method involves permuting each feature and measuring the decrease in model accuracy, thus indicating the importance of the feature. Using the feature importance scores from a random forest, one can select the top-N important features. This helps in reducing the dimensionality of the dataset by retaining only the most informative features.
- **Applications:**
  - **Data Visualization:** Reducing high-dimensional data to 2D or 3D for visualization.
  - **Noise Reduction:** Removing noise and redundant features from data.
  - **Speeding Up Algorithms:** Reducing the number of features to improve the efficiency of machine learning algorithms.
  - **Feature Extraction:** Identifying and creating new features that capture important information from the original data.

21. What is the difference between supervised and unsupervised learning? Give some examples of problems where we can use unsupervised learning.

Brief Description:
Supervised and unsupervised learning are two primary types of machine learning. The main difference lies in the presence of labeled data. Supervised learning uses labeled data to train models, while unsupervised learning uses unlabeled data to find patterns or structures within the data.

Detailed Description:

- **Supervised Learning:**
  - **Definition:** In supervised learning, the model is trained on a labeled dataset, which means each training example is paired with an output label. The goal is to learn a mapping from inputs to outputs that can be generalized to new, unseen data.
  - **Tasks:** Common tasks include classification and regression.
    - **Classification:** Predicting categorical labels (e.g., spam detection, image classification).
    - **Regression:** Predicting continuous values (e.g., predicting house prices, stock market forecasting).
  - **Example:** Predicting whether an email is spam or not based on its content and metadata. Here, the training dataset consists of emails labeled as "spam" or "not spam."
- **Unsupervised Learning:**

- **Definition:** In unsupervised learning, the model is trained on an unlabeled dataset. The goal is to find hidden patterns or intrinsic structures in the data without prior knowledge of the labels.
- **Tasks:** Common tasks include clustering, dimensionality reduction, and association.
  - **Clustering:** Grouping similar data points together (e.g., customer segmentation, document clustering).
  - **Dimensionality Reduction:** Reducing the number of features while preserving important information (e.g., Principal Component Analysis, t-SNE).
  - **Association:** Finding rules that describe large portions of the data (e.g., market basket analysis).

- **Examples of Problems Using Unsupervised Learning:**

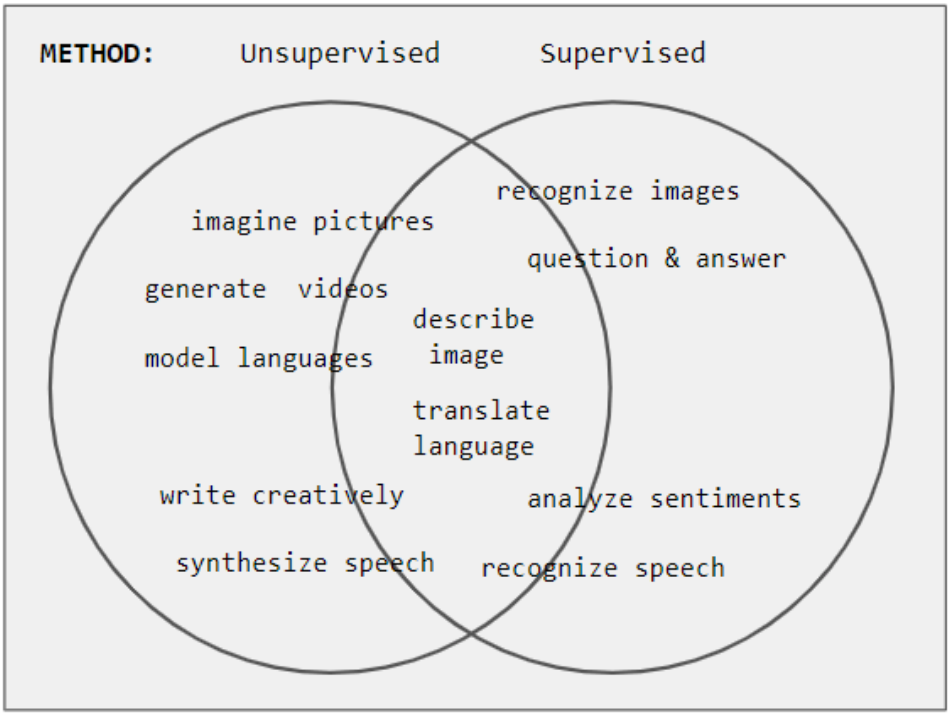
i. **Customer Segmentation:**
   - **Problem:** A retail company wants to group customers based on their purchasing behavior to tailor marketing strategies.
   - **Solution:** Use clustering algorithms like K-means or hierarchical clustering to segment customers into different groups with similar behavior.

ii. **Anomaly Detection:**
   - **Problem:** Identifying unusual patterns or outliers in network traffic that might indicate fraudulent activities or security breaches.
   - **Solution:** Use techniques like DBSCAN (Density-Based Spatial Clustering of Applications with Noise) or autoencoders to detect anomalies.

iii. **Document Clustering:**
   - **Problem:** Organizing a large collection of documents into meaningful clusters to facilitate information retrieval.
   - **Solution:** Use clustering algorithms like K-means or Latent Dirichlet Allocation (LDA) to group similar documents based on their content.

iv. **Dimensionality Reduction for Visualization:**
   - **Problem:** Visualizing high-dimensional data in a lower-dimensional space to understand the structure and relationships in the data.
   - **Solution:** Apply dimensionality reduction techniques like PCA or t-SNE to project the data into two or three dimensions for visualization.

v. **Market Basket Analysis:**
   - **Problem:** Understanding which products are frequently bought together to optimize product placement and promotions.
   - **Solution:** Use association rule learning algorithms like Apriori or FP-Growth to identify common itemsets and generate association rules.



Tendency for a task to employ supervised vs. unsupervised methods. Task names straddling circle boundaries is intentional. It shows that the classical division of imaginative tasks (left) employing unsupervised methods is blurred in today's learning schemes.

22. What problems may occur if we build too complex decision tree relative to the amount of data, and how to deal with them.

Brief Description:

Building a too complex decision tree relative to the amount of data can lead to overfitting, where the model captures noise in the training data rather than the underlying patterns. This results in poor generalization to new, unseen data. Various techniques can be used to prevent overfitting and improve the model's performance.

Detailed Description:

- **Problems with Too Complex Decision Trees:**
  - **Overfitting:** A complex decision tree can fit the training data perfectly, including noise and outliers, leading to poor performance on validation/test data.
  - **High Variance:** The model becomes sensitive to small variations in the training data, resulting in high variance and instability.
  - **Interpretability:** A complex tree can be difficult to interpret and understand, reducing its practical usefulness.
- **Techniques to Prevent Overfitting:**
  i. **Pruning:**

- **Definition:** Pruning involves removing parts of the tree that do not provide significant power in predicting target variables. There are two types of pruning:
  - **Pre-pruning (Early Stopping):** Stop growing the tree before it becomes too complex. Criteria include maximum depth, minimum number of samples required to split a node, and minimum number of samples per leaf.
  - **Post-pruning:** Remove branches from a fully grown tree that have little importance, based on a validation set or cost-complexity criteria.
- **Example:** Setting the maximum depth of the tree to a reasonable value or using cost-complexity pruning in libraries like scikit-learn.

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(max_depth=5)  # Pre-pruning by setting max depth
clf.fit(X_train, y_train)
```

ii. **Cross-Validation:**

- **Definition:** Use cross-validation techniques to evaluate the model's performance on different subsets of the data. This helps in tuning **hyperparameters** and selecting the best model that generalizes well.
- **Example:** Using k-fold cross-validation to determine the optimal tree depth.

```
from sklearn.model_selection import cross_val_score

clf = DecisionTreeClassifier(max_depth=5)
scores = cross_val_score(clf, X, y, cv=5)
print(scores.mean())
```

iii. **Ensemble Methods:**

- **Definition:** Combine multiple decision trees to form an ensemble model that is more robust and less prone to overfitting. Common ensemble methods include:
  - **Bagging (Bootstrap Aggregating):** Create multiple trees using bootstrap samples of the data and average their predictions (e.g., Random Forest).
  - **Boosting:** Build trees sequentially, where each tree corrects the errors of the previous ones (e.g., AdaBoost, Gradient Boosting).
- **Example:** Using Random Forest to reduce overfitting by averaging the predictions of multiple trees. (Clear ensamble method)

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)
```

iv. **Regularization:**

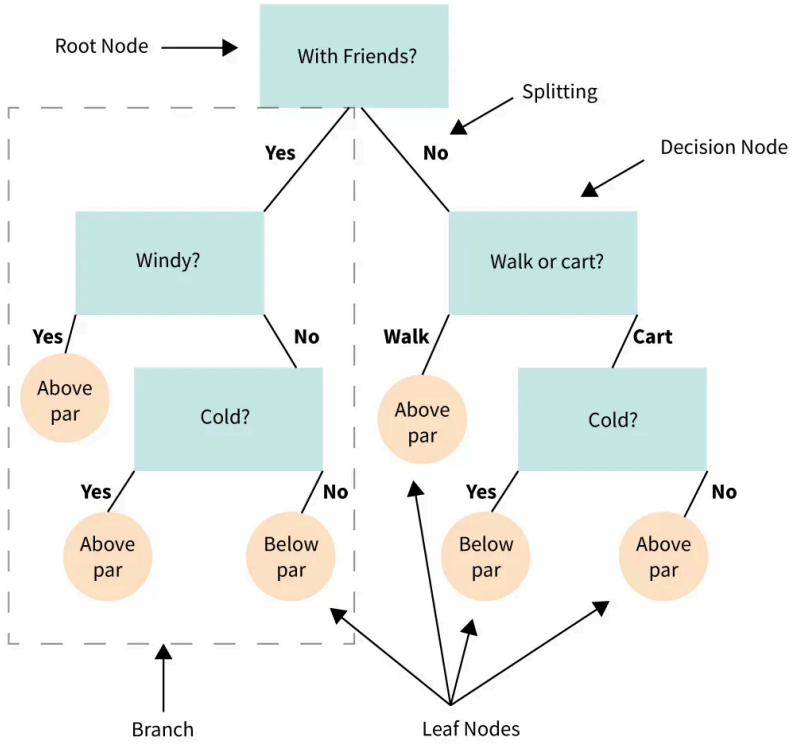
- **Definition:** Add a penalty term to the objective function to discourage overly complex models. Regularization techniques include L1 (Lasso) and L2 (Ridge) regularization.
- **Example:** Adding a complexity parameter to control the size of the tree.

```
clf = DecisionTreeClassifier(max_depth=5, min_samples_split=10)
clf.fit(X_train, y_train)
```

v. **Feature Selection:**

- **Definition:** Use feature selection techniques to remove irrelevant or redundant features, reducing the dimensionality of the data and simplifying the tree.
- **Example:** Using feature importance scores from a trained model to select the most important features.

```
feature_importances = clf.feature_importances_
selected_features = X_train.columns[feature_importances > threshold]
X_train_selected = X_train[selected_features]
clf.fit(X_train_selected, y_train)
```

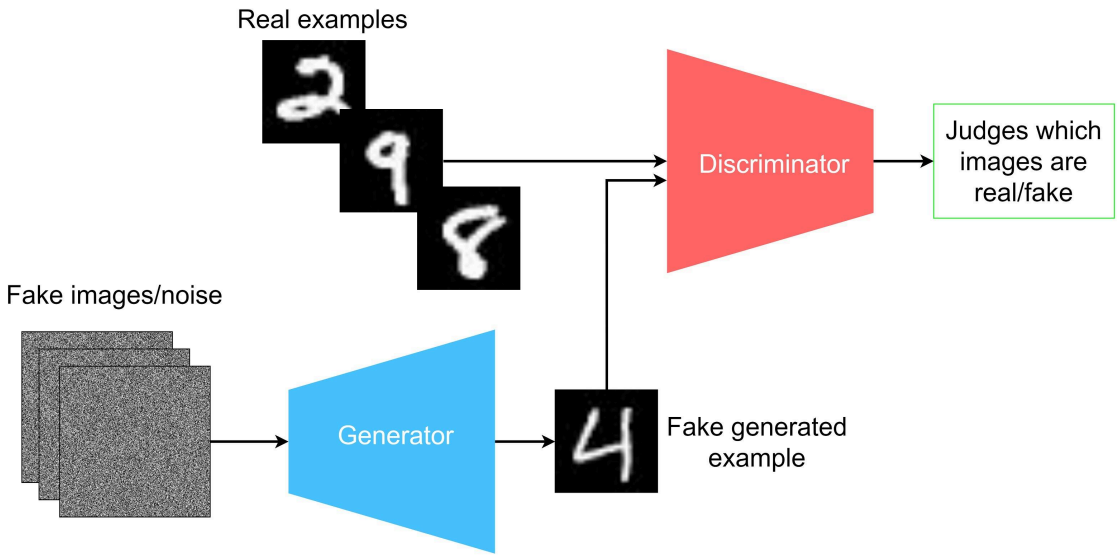


23. What is a generative adversarial network, and how to train such a network.

Brief Description:

A Generative Adversarial Network (GAN) is a class of machine learning frameworks designed by Ian

Goodfellow and his colleagues in 2014. It consists of two neural networks, a generator and a discriminator, which are trained together in a competitive process.



Detailed Description:

- **Structure of GANs:**
  i. **Generator (G):**
    - **Purpose:** The generator's goal is to create data that is similar to the real data. It takes random noise as input and generates synthetic data.
    - **Architecture:** Typically a neural network that maps a noise vector z to a data space $G(z)$.
  ii. **Discriminator (D):**
    - **Purpose:** The discriminator's goal is to distinguish between real data and the data generated by the generator. It takes an input and outputs a probability indicating whether the input is real or fake.
    - **Architecture:** A neural network that classifies inputs as real or fake, $D(x)$, where x is either real data or generated data.
- **Training Process:**
  The training of GANs involves a minimax game between the generator and the discriminator:
  - **Discriminator Training:** Update the discriminator to maximize the probability of correctly classifying real and generated data.

$$\max_D \ V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

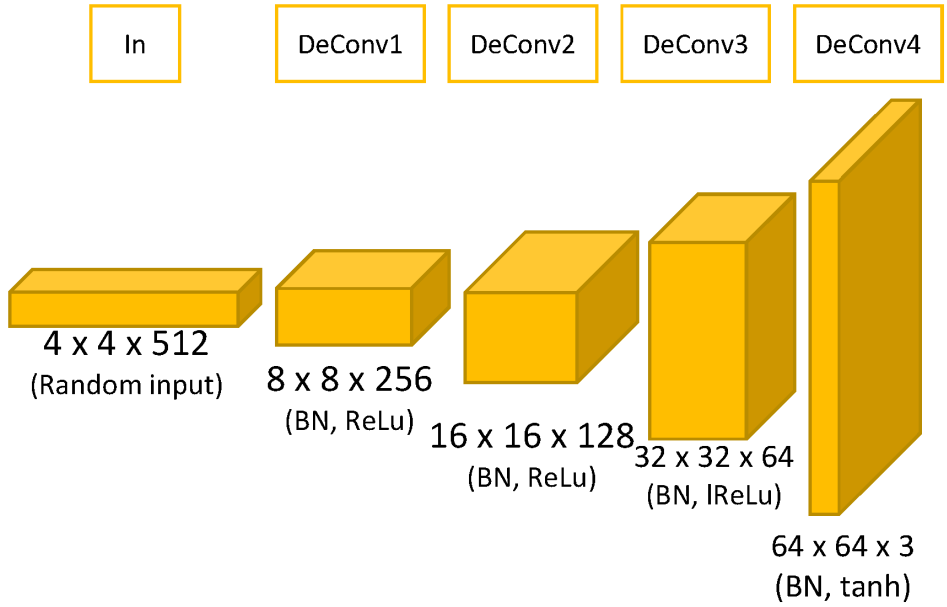
  - **Generator Training:** Update the generator to minimize the probability that the discriminator correctly identifies the generated data as fake.

$$\min_G \ V(D,G) = \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

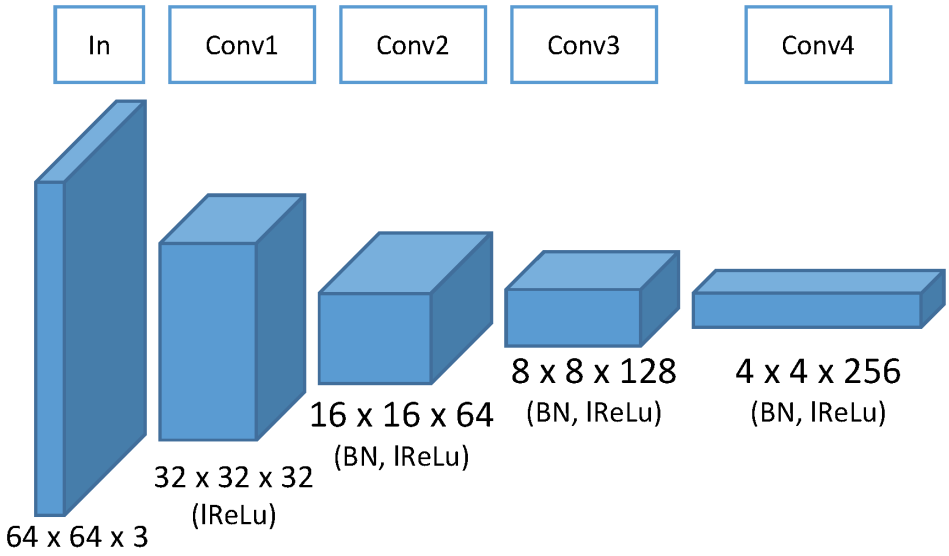
- **Alternating Optimization:**

i. **Step 1:** Train the discriminator on a batch of real data and a batch of data generated by the generator. Calculate the discriminator's loss and update its weights.
ii. **Step 2:** Train the generator using the feedback from the discriminator. Calculate the generator's loss and update its weights to improve the generation of realistic data.
iii. **Repeat:** Alternate between these two steps until the generator produces sufficiently realistic data, and the discriminator can no longer distinguish between real and fake data effectively.

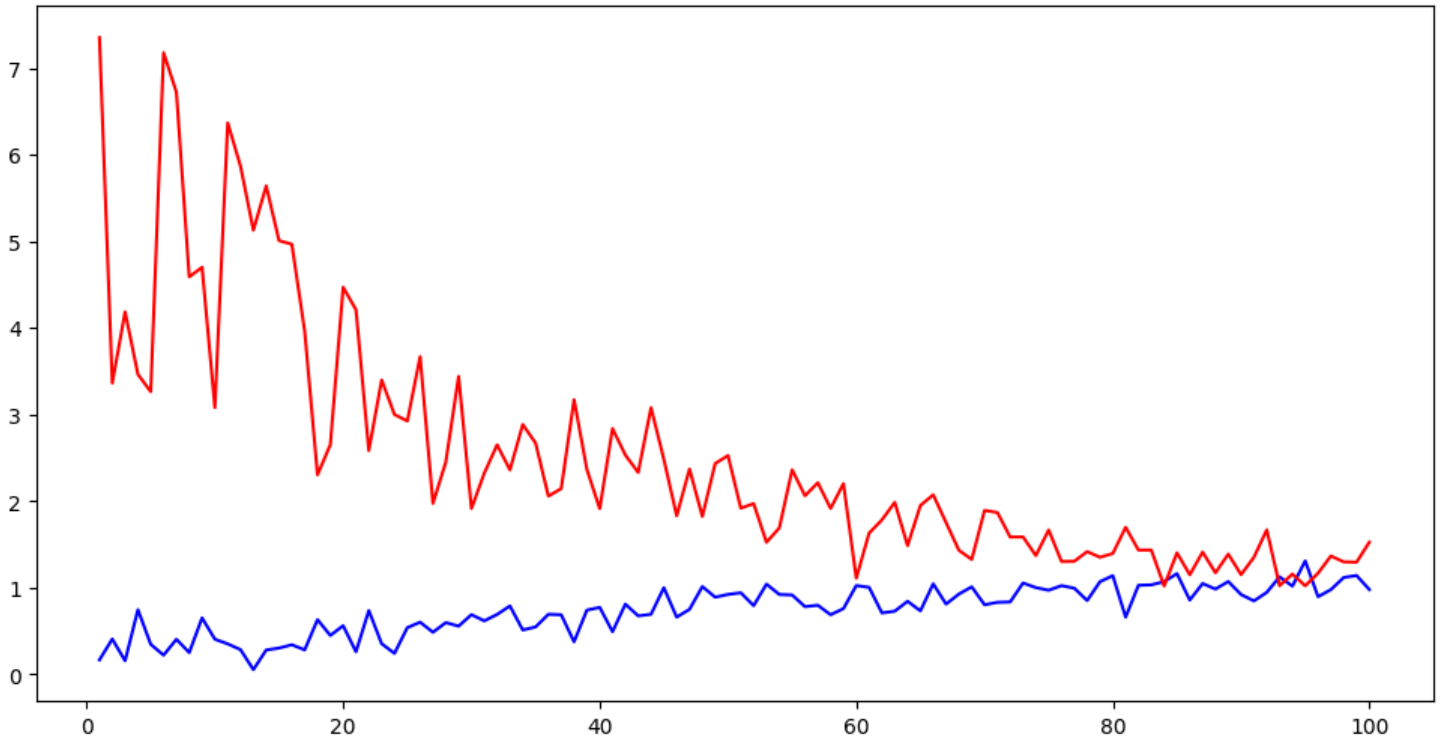
- **Generator (G):**



- **Discriminator (D):**



GAN training: red(generator), blue(discriminator)



24. Describe the family of exponential smoothing forecasting models.

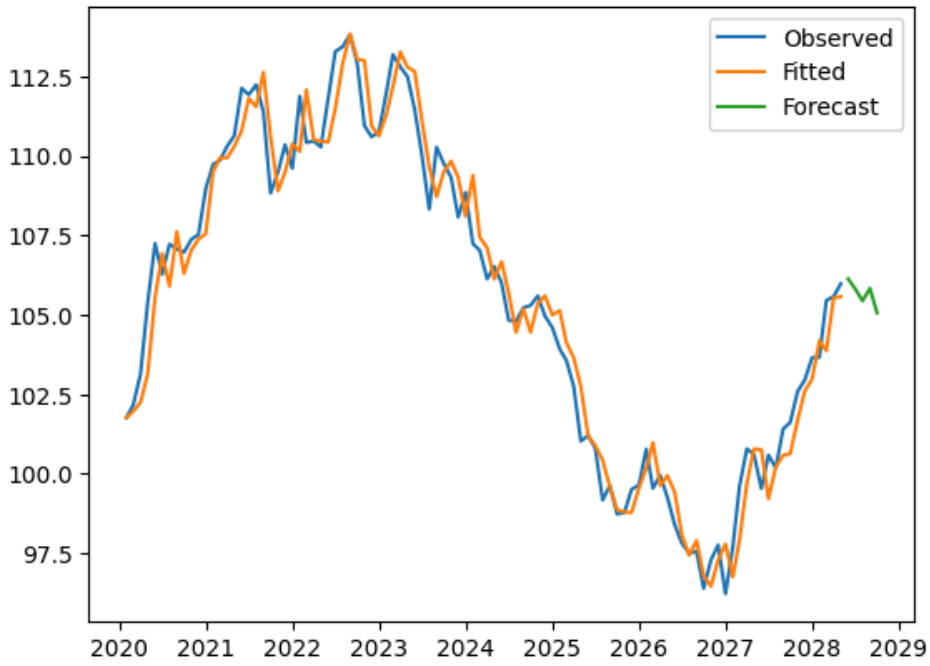
Brief Description:
Exponential smoothing forecasting models are a family of time series forecasting methods that apply weighted averages of past observations, with the weights decaying exponentially as the observations get older. These models are simple, robust, and effective for a wide range of time series data.

Detailed Description:

- **Simple Exponential Smoothing (SES):**
  - **Definition:** SES is used for forecasting time series data without trend or seasonality. It applies exponentially decreasing weights to past observations.
  - **Equation:** The forecast for time $t + 1$ is:
    $\hat{y}_{t+1} = \alpha y_t + (1 - \alpha)\hat{y}_t$
    where α is the smoothing parameter (0 < α < 1).
  - **Properties:** Suitable for level time series data without trends or seasonal patterns.
- **Holt's Linear Trend Model:**
  - **Definition:** Holt's model extends SES to capture linear trends in the data.
  - **Equations:**
    - Level equation:
      $l_t = \alpha y_t + (1 - \alpha)(l_{t-1} + b_{t-1})$

- Trend equation:
  $b_t = \beta(l_t - l_{t-1}) + (1 - \beta)b_{t-1}$
- Forecast equation:
  $\hat{y}_{t+h} = l_t + hb_t$
  where α and β are smoothing parameters for the level and trend, respectively.

- **Holt-Winters' Seasonal Model:**
  - **Definition:** Holt-Winters' model extends Holt's linear trend model to account for seasonality.
  - **Types:**
    - **Additive Seasonality:** Suitable for data with constant seasonal variations.
    - **Multiplicative Seasonality:** Suitable for data with seasonality that changes proportionally with the level.
  - **Equations (Additive):**
    - Level equation:
      $l_t = \alpha(y_t - s_{t-m}) + (1 - \alpha)(l_{t-1} + b_{t-1})$
    - Trend equation:
      $b_t = \beta(l_t - l_{t-1}) + (1 - \beta)b_{t-1}$
    - Seasonal equation:
      $s_t = \gamma(y_t - l_t) + (1 - \gamma)s_{t-m}$
    - Forecast equation:
      $\hat{y}_{t+h} = l_t + hb_t + s_{t-m+h}$
      where α, β, and γ are smoothing parameters for the level, trend, and seasonality, respectively, and m is the seasonal period.



25. Auto-regressive models in time series forecasting (ARMA, ARIMA, SARIMA, SARIMAX).

Brief Description:

Auto-regressive models are used for analyzing and forecasting time series data. They rely on the relationship between an observation and a number of lagged observations. The models extend from ARMA to ARIMA, SARIMA, and SARIMAX to handle various complexities in the data.

Detailed Description:

- **ARMA (AutoRegressive Moving Average):**
  - **Components:** Combines AutoRegressive (AR) and Moving Average (MA) models.
    - **AR(p) Model:** Uses past values (lags) to predict future values.
      $y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t$
    - **MA(q) Model:** Uses past forecast errors to predict future values.
      $y_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}$
    - **ARMA(p, q) Model:** Combines both AR and MA components.

$$y_t = \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q}$$

- **ARIMA (AutoRegressive Integrated Moving Average):**
  - **Components:** Extends ARMA to include differencing to make the time series stationary.
    - **Integrated (I):** Represents the differencing of observations to make the series stationary.
    - **ARIMA(p, d, q) Model:** Combines AR, I, and MA components.

$$y_t = \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q}$$

    - **Differencing:** d indicates the number of times the data has been differenced.

$$y'_t = y_t - y_{t-1}$$

- **SARIMA (Seasonal AutoRegressive Integrated Moving Average):**
  - **Components:** Extends ARIMA to handle seasonality.
    - **Seasonal Differencing:** Accounts for seasonality by differencing at the seasonal period.
    - **SARIMA(p, d, q)(P, D, Q)** Combines non-seasonal (ARIMA) and seasonal (SAR) components.

$$\begin{aligned} y_t = \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \Phi_1 Y_{t-m} + \cdots + \Phi_P Y_{t-Pm} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} \\ + \Theta_1 E_{t-m} + \cdots + \Theta_Q E_{t-Qm} \end{aligned}$$

    - **Parameters:**
      - p, d, q: Non-seasonal ARIMA parameters.
      - P, D, Q: Seasonal ARIMA parameters.
      - m: Seasonal period.
- **SARIMAX (Seasonal AutoRegressive Integrated Moving Average with eXogenous factors):**

- **Components:** Extends SARIMA to include exogenous variables (predictors) that can affect the forecast.
  - **SARIMAX(p, d, q)(P, D, Q) with Exogenous Regressors:** Adds external predictors to the SARIMA model.

$$y_t = \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \Phi_1 Y_{t-m} + \cdots + \Phi_P Y_{t-Pm} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} \\ + \Theta_1 E_{t-m} + \cdots + \Theta_Q E_{t-Qm} + \beta_1 X_{1t} + \cdots + \beta_k X_{kt}$$

  - **Exogenous Variables:** X_{kt} are the exogenous variables.
    Choosing parameters for those modesl requires lots of preperation and analysis of time series and model diagonstics.

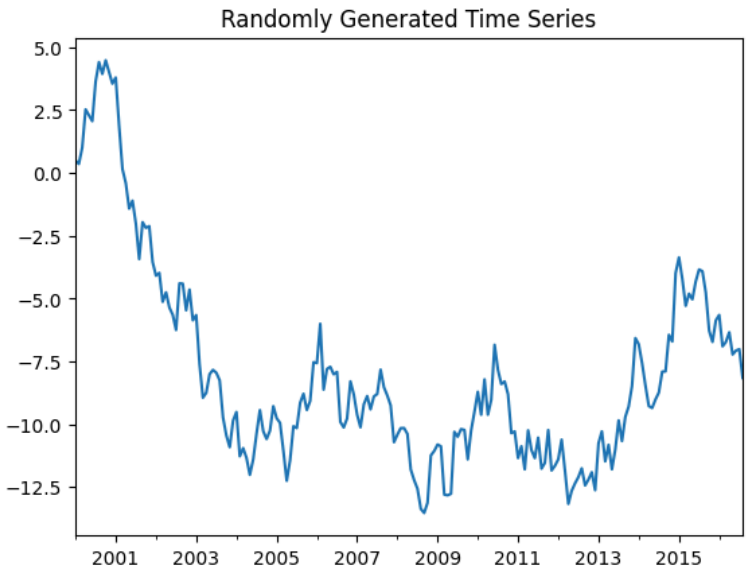
Example of the calculations in python using statsmodel:

- ## Data Generation:

A random time series dataset is generated using a cumulative sum of random values to introduce a trend.

- ## Visualization:

The time series is plotted to understand its structure.



- ## Stationarity Check:

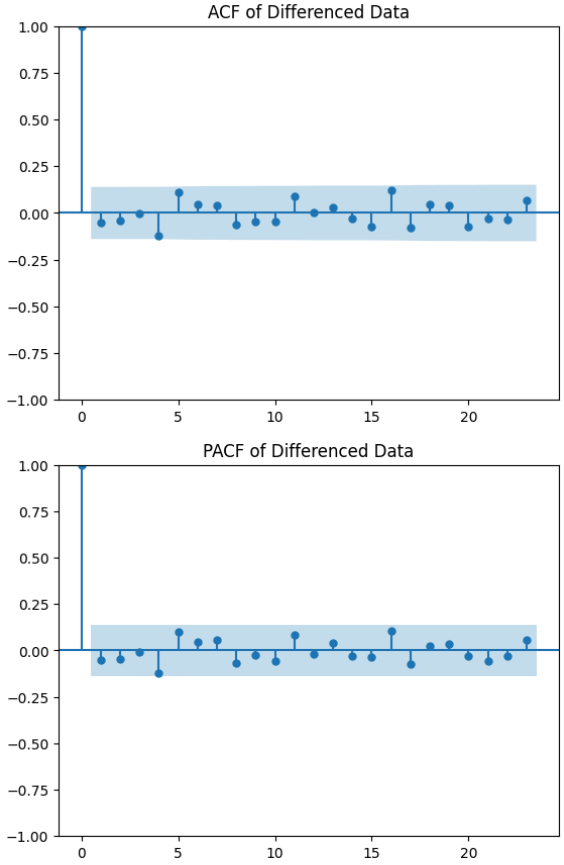
The Augmented Dickey-Fuller test is used to check if the series is stationary. If the series is not stationary, differencing is applied.

- ## ACF and PACF Plots:

  - **ACF (Autocorrelation Function):** The ACF measures the correlation between the time series and its lagged values. It helps identify the Moving Average (MA) component by showing how past

values of the series relate to current values.

- **PACF (Partial Autocorrelation Function):** The PACF measures the correlation between the time series and its lagged values, controlling for the values of the time series at all shorter lags. It helps identify the Autoregressive (AR) component by showing the direct relationship between past values and the current value after removing the influence of intermediate lags.



These plots help identify potential values for the AR (p) and MA (q) orders.

- ## Grid Search:

A grid search is performed over a range of potential parameter values to find the best combination based on the AIC criterion.

- ## Model Fitting:

The SARIMAX model is fitted using the best parameters found from the grid search.

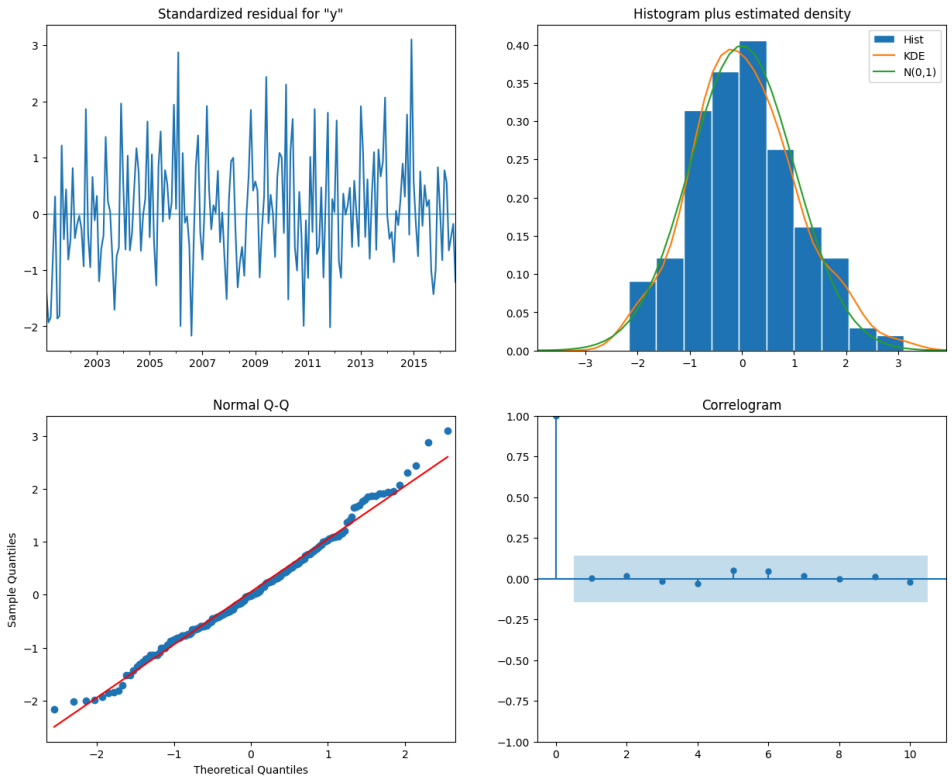
- ## Diagnostics:

Diagnostic plots are generated to validate the model. These plots include the residuals, ACF of residuals, and Q-Q plot.

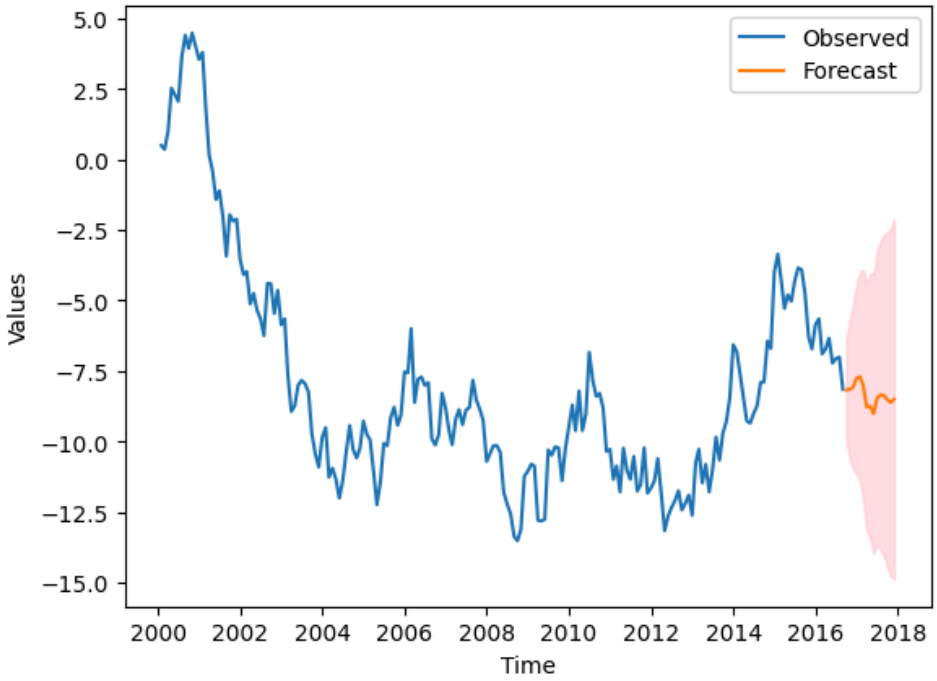
- **Model Summary:**

The summary of the fitted model is displayed, providing detailed information about the model parameters and their significance.

This process demonstrates how to apply SARIMAX to a time series dataset, including parameter selection and model diagnostics.



- **Model prediction with confidence intervals:**



26. Multi-layer perceptrons, recurrent and convolutional neural networks for time series forecasting.

Brief Description:

Multi-layer perceptrons (MLPs), recurrent neural networks (RNNs), and convolutional neural networks (CNNs) are three types of neural network architectures used for time series forecasting. Each has unique features and strengths that make them suitable for different types of time series data.

Detailed Description:

- **Multi-layer Perceptrons (MLPs):**
  - **Definition:** MLPs are feedforward neural networks consisting of an input layer, one or more hidden layers, and an output layer.
  - **Application to Time Series:**
    - MLPs can be applied to time series forecasting by using past time steps as features for predicting future values. This requires transforming the time series data into a supervised learning format.
  - **Strengths:**
    - Simple and straightforward to implement.
    - Can model non-linear relationships in the data.
  - **Limitations:**
    - Does not inherently capture temporal dependencies; requires explicit feature engineering to include lagged values.
- **Recurrent Neural Networks (RNNs):**
  - **Definition:** RNNs are designed to handle sequential data by maintaining a hidden state that captures information from previous time steps. Variants like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) are particularly effective for time series forecasting.
  - **Application to Time Series:**
    - RNNs process input sequences, making them well-suited for forecasting tasks where temporal dependencies are crucial.
  - **Strengths:**
    - Captures long-term dependencies and patterns in sequential data.
    - Suitable for complex time series with dynamic temporal behaviors.
  - **Limitations:**
    - Can be computationally intensive and require careful tuning to avoid issues like vanishing gradients.
- **Convolutional Neural Networks (CNNs):**
  - **Definition:** CNNs are designed to recognize spatial patterns using convolutional layers, which can be adapted to capture patterns in time series data by treating the temporal sequence as a

spatial sequence.

- **Application to Time Series:**
  - CNNs can be applied to time series forecasting by using 1D convolutions to detect local patterns over time.
- **Strengths:**
  - Effective at detecting local patterns and trends.
  - Can be combined with other architectures (e.g., ConvLSTM) for enhanced performance.
- **Limitations:**
  - May not capture long-term dependencies as effectively as RNNs

Comparison and Use Cases:

- **MLPs:** Suitable for simple time series without strong temporal dependencies, where feature engineering can create informative inputs.
- **RNNs (LSTM/GRU):** Ideal for time series with strong temporal dependencies, such as stock prices, weather data, and language modeling.
- **CNNs:** Useful for capturing local patterns in time series data, such as periodic trends or short-term fluctuations, and can be combined with RNNs for comprehensive modeling.

Add also here: encoder-decoder with attentions, transformers, graph neural networks.

"Still, in the area of pandemic prediction, there remain challenges for deep learning models: the time series is not long enough for effective training, unawareness of accumulated scientific knowledge, and interpretability of the model. To this end, the development of foundation models (large deep learning models with extensive pre-training) allows models to understand patterns and acquire knowledge that can be applied to new related problems before extensive training data becomes available."

SARIMAX is better when small amount of data is available! So some pre-trained models?

27. Methods of Analysis of Complex/Chaotic Behavior on the Basis of Famous Nonlinear Dynamics Models

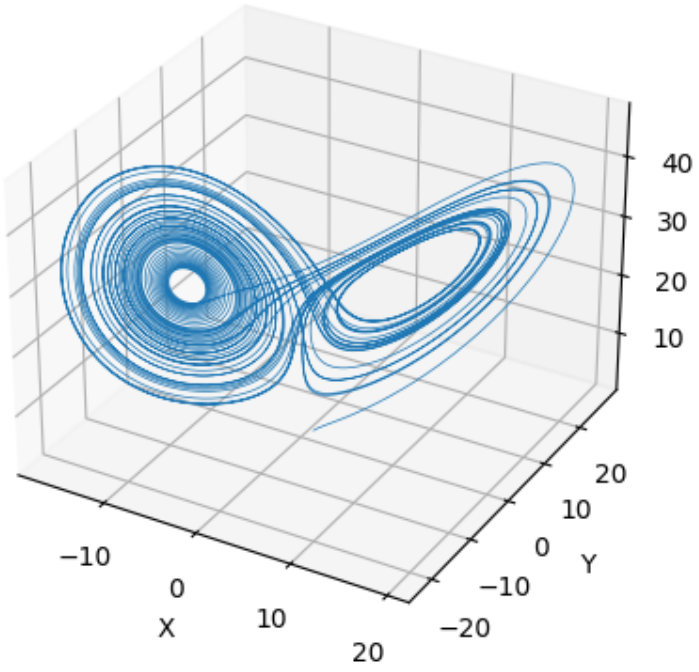
Brief Description:
The analysis of complex and chaotic behavior in nonlinear dynamical systems involves studying how small changes in initial conditions can lead to vastly different outcomes. Various methods and models, such as the Lorenz system, the logistic map, and phase space reconstruction, are used to understand and characterize this behavior. The double-rod pendulum is one of the simplest dynamical systems with chaotic solutions.

Detailed Description:

1. **Lorenz System:**
   - **Model:** A set of three differential equations originally developed to model atmospheric convection.

$$\begin{cases} \frac{dx}{dt} = \sigma(y - x) \\ \frac{dy}{dt} = x(\rho - z) - y \\ \frac{dz}{dt} = xy - \beta z \end{cases}$$

     where σ, ρ, and β are parameters.
   - **Analysis Methods:**
     - **Phase Space Plotting:** Visualize trajectories in a three-dimensional space to observe the butterfly-shaped attractor.
     - **Lyapunov Exponents:** Calculate to quantify the rate of separation of infinitesimally close trajectories, indicating chaotic behavior.



2. **Logistic Map:**
   - **Model:** A simple nonlinear difference equation used to model population dynamics.

$$x_{n+1} = r x_n (1 - x_n)$$

     where r is a parameter.
   - **Analysis Methods:**

    - **Bifurcation Diagram:** Plot the possible long-term values of x as r varies to observe periodic and chaotic regimes.
    - **Lyapunov Exponent:** Measure to determine the stability of the system and the presence of chaos.
3. **Phase Space Reconstruction:**
    - **Method:** Used to reconstruct the phase space of a dynamical system from a time series of observations.
    - **Takens' Theorem:** Provides a framework for reconstructing a high-dimensional phase space from a single-dimensional time series using delay coordinates.

$$\mathbf{X}(t) = [x(t), x(t+\tau), x(t+2\tau), \ldots, x(t+(m-1)\tau)]$$

      where τ is the time delay and m is the embedding dimension.
    - **Applications:**
        - **Attractor Reconstruction:** Visualize the underlying attractor of the system.
        - **Predictability Analysis:** Study the system's sensitivity to initial conditions.
4. **Poincaré Map:**
    - **Method:** A technique to reduce the dimensionality of a continuous dynamical system by taking a cross-section of the phase space.
    - **Applications:**
        - **Periodic Orbits:** Identify and analyze periodic orbits and their stability.
        - **Chaos Detection:** Detect chaotic behavior through the complex structure of the Poincaré map.
5. **Fractal Dimension:**
    - **Method:** Measure to quantify the complexity of an attractor in phase space.
    - **Common Techniques:**
        - **Box-Counting Dimension:** Count the number of boxes of a certain size needed to cover the attractor.
        - **Correlation Dimension:** Estimate the fractal dimension based on the scaling of the correlation sum with distance.
6. **Recurrence Plot:**
    - **Method:** Visual tool to identify times at which a dynamical system returns to a previous state.
    - **Applications:**
        - **Recurrence Quantification Analysis (RQA):** Extract quantitative measures of system behavior, such as recurrence rate, determinism, and laminarity.
7. **Nonlinear Oscillators and Bifurcations:**
    - **Model:** Investigate systems like the Duffing oscillator, Van der Pol oscillator, and other nonlinear oscillators.

    - **Bifurcation Analysis:** Study how the qualitative nature of the system changes as parameters vary, identifying phenomena such as Hopf bifurcations, period-doubling, and chaotic regimes.
8. **Deterministic Chaos and Strange Attractors:**
    - **Model:** Explore systems that exhibit deterministic chaos, where the system's long-term behavior is highly sensitive to initial conditions.
    - **Examples:** Double pendulum, Rössler attractor, and the Chua circuit.
    - **Strange Attractors:** Analyze attractors that have a fractal structure and are associated with chaotic dynamics.
9. **Lyapunov Exponents and Stability Analysis:**
    - **Method:** Calculate Lyapunov exponents to measure the rates of separation of infinitesimally close trajectories.
    - **Stability:** Determine the stability of fixed points, periodic orbits, and chaotic attractors using Lyapunov exponents.
10. **Numerical Methods and Simulation Tools:**
    - **Software:** Use tools like Maple, MATLAB, and Python to simulate and analyze nonlinear dynamical systems.
    - **Techniques:** Employ numerical integration methods, such as the Runge-Kutta method, to solve differential equations and explore system dynamics.

Lyapunov Exponent

The Lyapunov exponent is a measure used in dynamical systems to characterize the rate of separation of infinitesimally close trajectories. It quantifies the sensitivity to initial conditions, which is a hallmark of chaotic systems.

Definition

The Lyapunov exponent, λ, is defined as:

$$\lambda = \lim_{t \to \infty} \lim_{\delta(0) \to 0} \frac{1}{t} \ln \frac{\delta(t)}{\delta(0)}$$

where:

- $\delta(0)$ is the initial separation between two trajectories in phase space.
- $\delta(t)$ is the separation between these trajectories at time t.

Interpretation

- **Positive Lyapunov Exponent**: Indicates chaos. Trajectories diverge exponentially fast, implying that small differences in initial conditions lead to vastly different outcomes.

- **Zero Lyapunov Exponent**: Indicates neutral stability, often associated with periodic or quasi-periodic orbits.
- **Negative Lyapunov Exponent**: Indicates stability. Trajectories converge, implying that the system tends to return to its initial state after small perturbations.

Calculation

1. **Numerical Method**:
   - Numerical simulations of the system's equations are used to follow the evolution of two initially close points in phase space.
   - The separation between these points is monitored over time, and the Lyapunov exponent is computed from the average exponential rate of divergence.
2. **Analytical Method**:
   - For some simple systems, Lyapunov exponents can be calculated analytically by linearizing the system around a fixed point and analyzing the eigenvalues of the Jacobian matrix.

Example: Logistic Map

The logistic map is a simple, well-known system that can exhibit chaotic behavior:

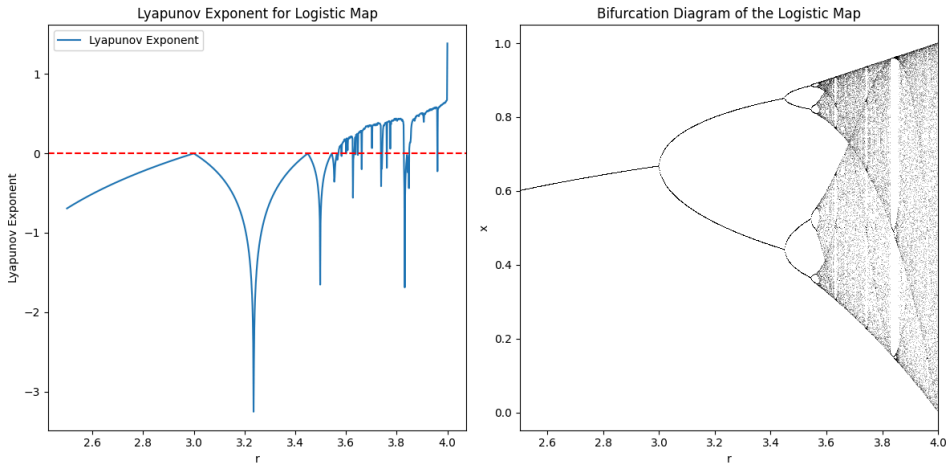
$$x_{n+1} = r x_n (1 - x_n)$$

To compute the Lyapunov exponent for the logistic map:

1. Iterate the map for a large number of steps.
2. Compute the average rate of divergence.

The formula for the Lyapunov exponent λ in this case is:

$$\lambda = \lim_{N \to \infty} \frac{1}{N} \sum_{n=1}^{N} \ln |r(1 - 2x_n)|$$



28. What is the Monte Carlo (MC) method? What mathematical theorem is related to MC? Metropolis algorithm. Analysis of MC data for Ising model

Brief Description:

The Monte Carlo (MC) method is a computational algorithm that uses repeated random sampling to obtain numerical results. It is often used to model complex systems and processes that are difficult to analyze analytically.

Detailed Description:

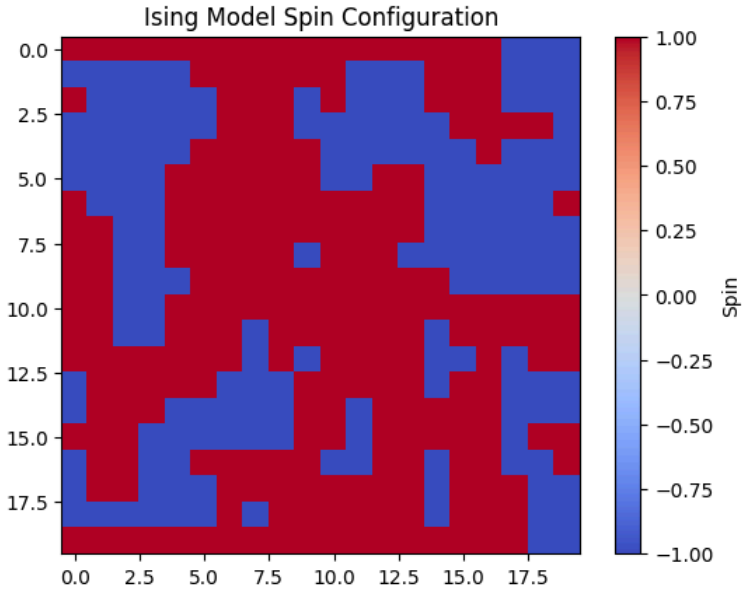
- **Monte Carlo (MC) Method:**
  - **Definition:** The Monte Carlo method involves using randomness to solve problems that might be deterministic in principle. By running simulations with random variables, it estimates mathematical functions and models the probability of different outcomes.
  - **Applications:** Widely used in physics, finance, engineering, and many other fields for integration, optimization, and probabilistic modeling.
- **Related Mathematical Theorem:**
  - **Law of Large Numbers:** The Monte Carlo method relies on the law of large numbers, which states that as the number of trials increases, the average of the results obtained from the trials converges to the expected value.
  - **Central Limit Theorem:** This theorem is also related, as it provides the foundation for the distribution of the sum of a large number of independent, identically distributed random variables, which converges to a normal distribution.
- **Metropolis Algorithm:**
  - **Definition:** The Metropolis algorithm is a Monte Carlo method used for obtaining a sequence of random samples from a probability distribution for which direct sampling is difficult. It is the basis for the more general Metropolis-Hastings algorithm.
  - **Steps:**
    a. **Initialization:** Start with an initial state x_0.
    b. **Proposal:** Generate a candidate state x' from a proposal distribution $q(x'|x)$.
    c. **Acceptance Probability:** Calculate the acceptance probability:
    $A(x \to x') = \min\left(1, \frac{P(x')q(x|x')}{P(x)q(x'|x)}\right)$
    d. **Acceptance/Rejection:** Accept the new state x' with probability $A(x \to x')$; otherwise, retain the current state x.
    e. **Iteration:** Repeat the proposal and acceptance steps for a large number of iterations.
  - **Usage:** The Metropolis algorithm is used to sample from complex distributions, especially in statistical mechanics and Bayesian statistics.
- **Analysis of MC Data for Ising Model:**
  - **Ising Model:**

- **Definition:** The Ising model is a mathematical model of ferromagnetism in statistical mechanics. It consists of discrete variables called spins, which can be in one of two states ($+1$ or -1). The spins are arranged on a lattice, and each spin interacts with its neighbors.
- **Hamiltonian:** The energy of a configuration is given by the Hamiltonian:

$$H = -J \sum_{\langle i,j \rangle} S_i S_j - h \sum_i S_i$$

where J is the interaction energy, S_i is the spin at site i, $\langle i, j \rangle$ indicates summation over nearest neighbors, and h is an external magnetic field.

- **Using Metropolis Algorithm for Ising Model:**
  a. **Initialization:** Start with a random spin configuration.
  b. **Monte Carlo Step:**
     - Select a spin randomly and flip it.
     - Calculate the change in energy ΔE due to the flip.
     - If $\Delta E \leq 0$, accept the flip. If $\Delta E > 0$, accept the flip with probability $\exp(-\Delta E / k_B T)$, where T is the temperature and k_B is the Boltzmann constant.
  c. **Iteration:** Repeat the Monte Carlo steps for many iterations to reach equilibrium.
  d. **Measurement:** After equilibration, measure physical quantities like magnetization, susceptibility, and specific heat.
- **Analysis:**
  - **Magnetization:** Average spin value, which indicates the overall magnetic moment. $M = \frac{1}{N} \sum_i S_i$
  - **Susceptibility:** Measures the response of magnetization to the external magnetic field. $\chi = \frac{\partial M}{\partial h}$
  - **Specific Heat:** Measures the response of the energy to temperature changes. $C = \frac{\partial E}{\partial T}$



29. Pseudo-random numbers and generators. Desired features of pseudo-random generators for Monte Carlo simulations. Generation of pseudo-random numbers with various probability density functions.

Brief Description:

Pseudo-random numbers are numbers that appear random but are generated by a deterministic algorithm. These numbers are crucial for simulations, cryptographic applications, and various algorithms that rely on random sampling.

Detailed Description:

- **Pseudo-random Numbers and Generators:**
  - **Definition:** Pseudo-random numbers are generated by algorithms that produce sequences of numbers approximating the properties of random sequences. These algorithms are called pseudo-random number generators (PRNGs).
  - **Common PRNG Algorithms:**
    - **Linear Congruential Generator (LCG):**
      $X_{n+1} = (aX_n + c) \mod m$
      where a, c, and m are integers that define the generator.
    - **Mersenne Twister:** A widely used PRNG known for its long period and high quality of randomness.
    - **Xorshift:** A family of PRNGs that use bitwise operations to produce random numbers efficiently.
- **Desired Features of Pseudo-random Generators for Monte Carlo Simulations:**
  i. **Uniform Distribution:** The numbers generated should be uniformly distributed over the desired range.

ii. **Independence:** Successive numbers should be independent of each other.
iii. **Long Period:** The generator should have a long period before the sequence repeats to ensure a wide range of values.
iv. **Reproducibility:** The ability to reproduce the same sequence of random numbers given the same initial seed.
v. **Efficiency:** The generator should produce numbers quickly with minimal computational overhead.
vi. **Portability:** The generator should be consistent across different platforms and programming environments.

- **Generation of Pseudo-random Numbers with Various Probability Density Functions:**
  - **Transformations from Uniform Distribution:** Many techniques transform uniformly distributed random numbers into numbers that follow different probability distributions.
  - **Common Methods:**
    a. **Inverse Transform Sampling:** Use the inverse of the cumulative distribution function (CDF) to transform a uniform random number into a number following the desired distribution.
    $X = F^{-1}(U)$
    where U is a uniform random variable, F is the CDF of the desired distribution, and F^{-1} is its inverse.
    b. **Box-Muller Transform:** Generates normally distributed random numbers from uniformly distributed random numbers.
    $Z_0 = \sqrt{-2 \ln U_1} \cos(2\pi U_2)$
    $Z_1 = \sqrt{-2 \ln U_1} \sin(2\pi U_2)$
    where U_1 and U_2 are independent uniform random numbers.
    c. **Rejection Sampling:** Generates random samples from any distribution by using a proposal distribution that envelopes the target distribution.
    d. **Ziggurat Algorithm:** Efficiently generates random numbers from a variety of distributions, including normal and exponential distributions.

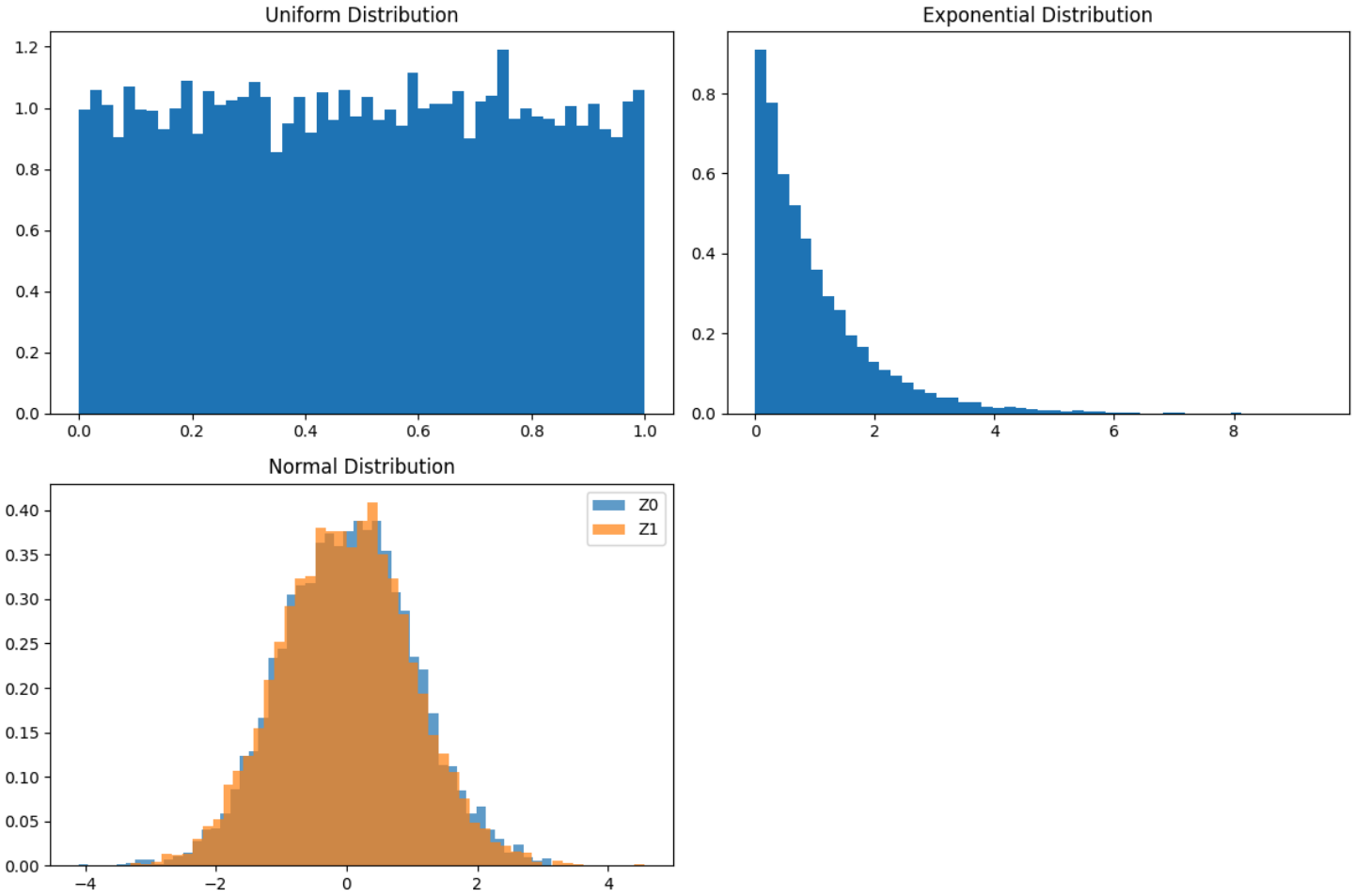
Example in Python:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Uniform random numbers
uniform_random_numbers = np.random.uniform(0, 1, 10000)

# Inverse transform sampling for exponential distribution (lambda=1)
exponential_random_numbers = -np.log(1 - uniform_random_numbers)

# Box-Muller transform for normal distribution
U1 = np.random.uniform(0, 1, 5000)
U2 = np.random.uniform(0, 1, 5000)
Z0 = np.sqrt(-2 * np.log(U1)) * np.cos(2 * np.pi U2)
Z1 = np.sqrt(-2 * np.log(U1)) * np.sin(2 * np.pi U2)
```



Further Reading:

"Numerical Recipes: The Art of Scientific Computing" by William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery.

30. Markov chains and their applications in statistical physics

Brief Description:

Markov chains are mathematical systems that transition from one state to another within a finite or countable state space. They are characterized by the Markov property, where the probability of transitioning to any particular state depends only on the current state and not on the previous states.

Detailed Description:

- **Markov Chains:**
  - **Definition:** A Markov chain is a stochastic process that undergoes transitions from one state to another on a state space. It is defined by a set of states and transition probabilities between these states.
  - **Markov Property:** The next state depends only on the current state and not on the sequence of events that preceded it.

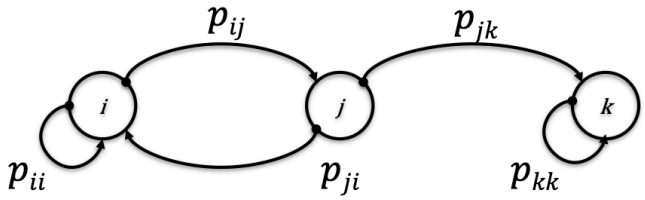
$$P(X_{n+1} = x | X_n = x_n, X_{n-1} = x_{n-1}, \ldots, X_0 = x_0) = P(X_{n+1} = x | X_n = x_n)$$

- **Transition Matrix:**
  - **Definition:** The transition probabilities are often represented in a matrix form, where the element P_{ij} represents the probability of transitioning from state i to state j.

$$P = \begin{pmatrix} P_{11} & P_{12} & \cdots & P_{1n} \\ P_{21} & P_{22} & \cdots & P_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n1} & P_{n2} & \cdots & P_{nn} \end{pmatrix}$$

  - **Properties:**
    - Each row sums to 1: $\sum_j P_{ij} = 1$
    - Elements are non-negative: $0 \leq P_{ij} \leq 1$

This transition matrix can be described also using a directed graph



Note: different values of the same random variable!

Note: this describes how a random variable changes during time!

Stochastic Processes and Markov Chains

Given X_t the value of a (state) random variable at time t :

- *Discrete Stochastic Process* describes the relationship between the stochastic description of a system $(X_0, X_1, X_2, \dots)$ at some discrete time steps.
- *Continuous Stochastic Process* is a stochastic process where the state can be observed at any time.

A Discrete Stochastic Process is a (first order) *Markov Chain* when we have that $\forall t = 1,2,3, \dots$ and for all N states it holds:

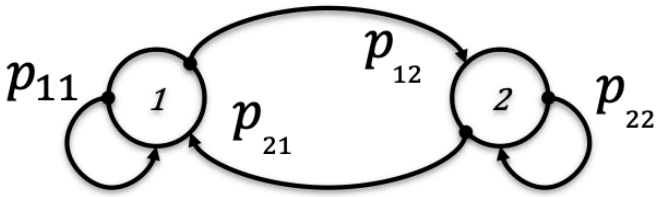
$$P(X_t|X_{t-1}, X_{t-2}, .., X_0) = P(X_t|X_{t-1})$$

Whenever the probability of an event is independent from time the Markov Chain is *Stationary*: $P(X_{t+1} = j|X_t = i) = p_{ij}$

If all states in a Markov Chain are recurrent, a-periodic, and communicate with each other, it is said to be Ergodic

to compute a probablity of a stationary markov chain over t time you need to multiply tranistion matrix t times !

$$P = \begin{array}{c} \\ Cola_1 \\ Cola_2 \end{array} \begin{array}{c} \begin{array}{cc} Cola_1 & Cola_2 \end{array} \\ \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} \end{array}$$



Someone has bought $\boldsymbol{Cola_2}$, how likely she'll buy $\boldsymbol{Cola_1}$ after 2 times?

$$P(X_2 = 1|X_0 = 2) = P_{21}\ (2)$$

$$P(2) = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}\begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} = \begin{bmatrix} 0.83 & 0.17 \\ 0.34 & 0.66 \end{bmatrix}$$

Someone has bought $\boldsymbol{Cola_1}$, how likely she'll buy $\boldsymbol{Cola_1}$ again after 3 times?

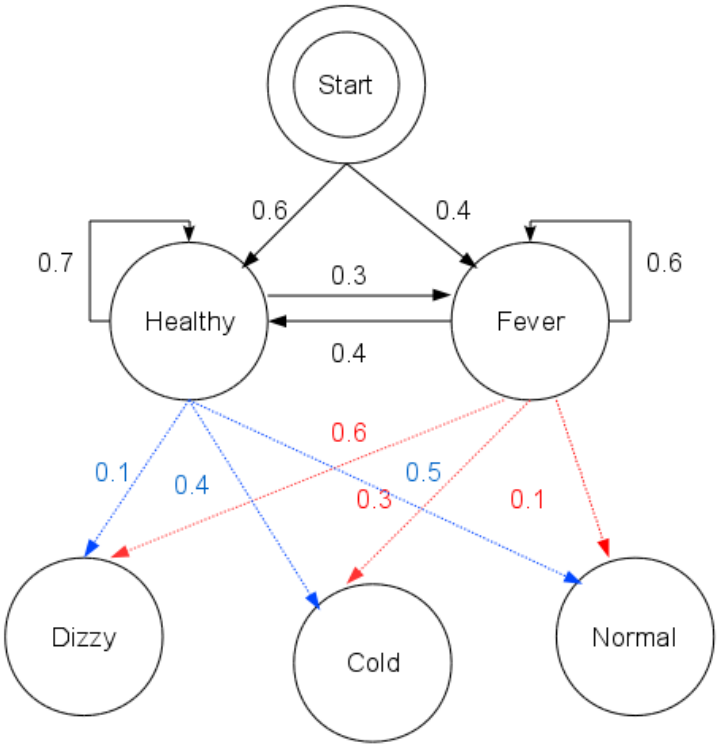
$$P(X_3 = 1|X_0 = 1) = P_{11}\ (3)$$

$$P(3) = \begin{bmatrix} 0.83 & 0.17 \\ 0.34 & 0.66 \end{bmatrix}\begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} = \begin{bmatrix} 0.781 & 0.219 \\ 0.438 & 0.562 \end{bmatrix}$$

- **Applications in Statistical Physics:**
  - **Monte Carlo Simulations:**
    - Markov chains are used in Monte Carlo simulations to sample from complex probability distributions. The Metropolis algorithm and Metropolis-Hastings algorithm are examples where Markov chains are employed to explore the state space.
  - **Modeling Thermodynamic Systems:**
    - Markov chains model the behavior of particles in a thermodynamic system, where each state represents a possible configuration of the system. This helps in understanding phenomena like phase transitions and equilibrium states.
  - **Ising Model:**
    - The Ising model, a mathematical model of ferromagnetism in statistical mechanics, uses Markov chains to simulate the spin configurations of particles and study their magnetic properties.
  - **Diffusion Processes:**
    - Markov chains model diffusion processes, where particles move randomly, and their future positions depend only on their current positions. This is used to study phenomena like Brownian motion.
- **Example: Metropolis Algorithm in Statistical Physics:**
  The Metropolis algorithm is used to generate a sequence of states for a system with a given energy function, facilitating the study of thermodynamic properties.
- **Hidden markov state:**

The observations (normal, cold, dizzy) along with the hidden states (healthy, fever) form a hidden Markov model (HMM). From past experience, the probabilities of this model have been estimated as:

```
init = {"Healthy": 0.6, "Fever": 0.4}
trans = {
    "Healthy": {"Healthy": 0.7, "Fever": 0.3},
    "Fever": {"Healthy": 0.4, "Fever": 0.6},
}
emit = {
    "Healthy": {"normal": 0.5, "cold": 0.4, "dizzy": 0.1},
    "Fever": {"normal": 0.1, "cold": 0.3, "dizzy": 0.6},
}
```



Graphical representation of the given HMM

The Viterbi algorithm is a dynamic programming algorithm for obtaining the maximum a posteriori probability estimate of the most likely sequence of hidden states—called the Viterbi path—that results in a sequence of observed events. This is done especially in the context of Markov information sources and hidden Markov models (HMM).