Overview

https://www.mermaidchart.com/raw/886e353c-e7eb-4dd9-8e28-1d665972b951?theme=light&version=v0.1&format=svg

1. Artificial Intelligence (AI)
   ├── 1.1 Machine Learning (ML)
   │   ├── 1.1.1 Supervised Learning -> Prediction
   │   │   ├── 1.1.1.1 Regression
   │   │   │   ├── 1.1.1.1.1 Linear Regression
   │   │   │   ├── 1.1.1.1.2 Polynomial Regression
   │   │   ├── 1.1.1.2 Classification
   │   │   │   ├── 1.1.1.2.1 Binary Classification
   │   │   │   ├── 1.1.1.2.2 Multi-Class Classification
   │   │
   │   │
   │   ├── 1.1.2 Unsupervised Learning
   │   │   ├── 1.1.2.1 Clustering
   │   │   ├── 1.1.2.2 Dimensionality Reduction
   │   │   │   ├── 1.1.2.2.1 Principal Component Analysis (PCA)
   │   │
   │   ├── 1.1.3 Semi-Supervised Learning
   │   ├── 1.1.4 Reinforcement Learning
   │
   ├── 1.2 Specialized Algorithms and Techniques
   │   ├── 1.2.1 PageRank (Ranking Algorithms)
   │   ├── 1.2.2 Matrix Factorization
   │   │   ├── 1.2.2.1 Applications in Recommendation Systems
   │   │   ├── 1.2.2.2 Non-Negative Matrix Factorization (NMF)
   │   │   ├── 1.2.2.3 Singular Value Decomposition (SVD)
   │
   ├── 1.3 Data Science Techniques (Supporting AI)
   │   ├── 1.3.1 Feature Engineering
   │   ├── 1.3.2 Data Cleaning
   │   ├── 1.3.3 Exploratory Data Analysis (EDA)
   │   1.3.4 Feature Selection
   │
   ├── 1.4 Supporting Mathematical Foundations
   │   ├── 1.4.1 Linear Algebra
   │   │   ├── 1.4.1.1 Matrices and Matrix Operations
   │   │   ├── 1.4.1.2 Eigenvalues and Eigenvectors
   │   ├── 1.4.2 Probability and Statistics
   │   │   ├── 1.4.2.1 Bayesian Statistics

```
│   │   ├── 1.4.2.2 Hypothesis Testing
│   ├── 1.4.3 Optimization Techniques
        1.4.3.1 Gradient descent
│
├── 1.5 Applications of AI
    ├── 1.5.1 Natural Language Processing (NLP)
    ├── 1.5.2 Computer Vision: Images
    ├── 1.5.3 Robotics
    ├── 1.5.4 Recommendation Systems
    │   ├── 1.5.4.1 Collaborative Filtering
    │   ├── 1.5.4.2 Content-Based Filtering
        1.5.5  LLM (large language model): chatgpt, notebook llm
```

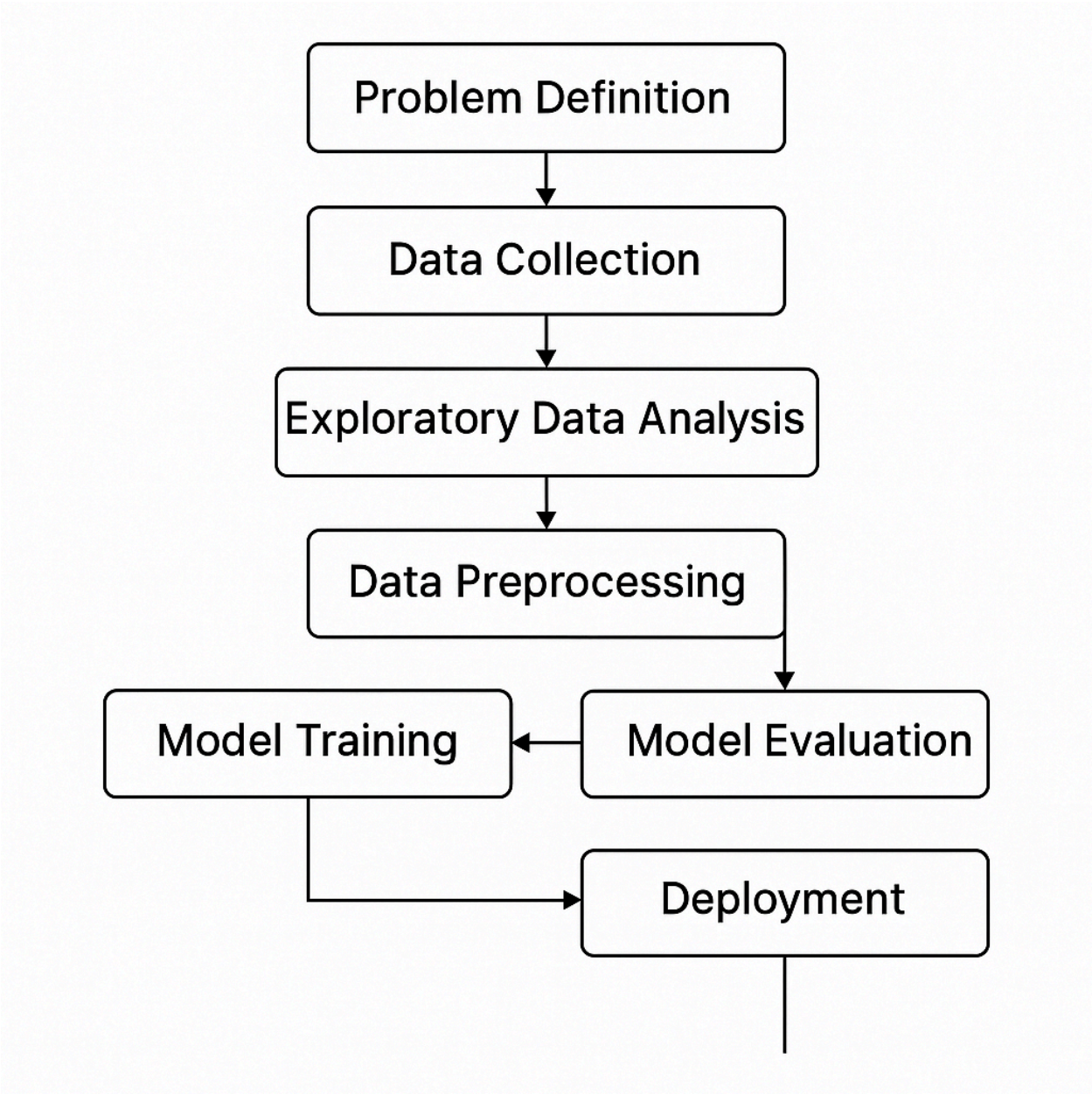
Syllabus

Module A
Basics of Python and Inspiration of Functions
Variables and Data Types
Loops and Conditions
Demonstration of File Handling
Taking inputs from users
Basics of File Handling
Tuple, List and Dictionaries
Algorithm Efficiency using function visualisation
Basics of Sorting (Bubble, Quick and Insertion)
Searching and it's relevance
Vector and Matrices (Operations)
Matrix Operations
Probability
Descriptive statistics - Mean, Median, Mode, Std dev, variance, range
Numpy
Pandas
Visualisation

Module B
Regression
Polynomial Regression
Dimensionality Reduction
Page Rank
Foundations of the Perceptron
Enhancing the Perceptron with Learning Algorithms
Neural networks - Forward and backward propagation

CNN
SVM
Supervised Learning - Evaluation metrics
Unsupervised Learning - K Means clustering
Unsupervised Learning - Evaluation metrics

ML Project Life cycle



Common Terms

Train-test-split

Scenario 1 (test_size=0.2):

- Total samples = 2000
- Test set = 20% of total (400 samples)
- Training set = 80% of total (1600 samples)

Scenario 2 (train_size=0.2):

- Total samples = 200
- Training set = 20% of total (40 samples)
- Test set = 80% of total (160 samples)

1. Error Metrics

- **Mean Absolute Error (MAE)**: Average of absolute differences between actual and predicted values.
  - Formula: $$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$
  - Measures overall error magnitude.
- **Mean Squared Error (MSE)**: Average of squared differences between actual and predicted values.
  - Formula: $$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
  - Penalizes larger errors more heavily than MAE.

---

2. Bias-Variance Tradeoff

- **Bias**: The error due to overly simplistic assumptions in the model. High bias leads to underfitting.
- **Variance**: The error due to sensitivity to small fluctuations in the training set. High variance leads to overfitting.

---

3. Model Complexity

- **Underfitting**: When the model is too simple to capture the underlying patterns in the data.
- **Overfitting**: When the model is too complex and captures noise in the data along with the signal.

---

4. Cross-Validation

- **K-Fold Cross-Validation**: Splitting the dataset into kkk parts and using each part as a test set once while training on the rest.
- **Train-Test Split**: Dividing the dataset into training and testing subsets to evaluate model performance.

---

6. Regularization (for Overfitting)

- **L1 Regularization (Lasso)**: Adds a penalty proportional to the absolute value of coefficients.
- **L2 Regularization (Ridge)**: Adds a penalty proportional to the squared value of coefficients.

---

7. Overfitting Indicators

- **Low Training Error, High Test Error**: A clear sign of overfitting.
- **Validation Curve**: Plot showing error across different model complexities. A U-shaped validation curve indicates overfitting at high complexities.

---

8. Feature Importance

- **Feature Contribution**: How much each feature contributes to the model's prediction.
- Helps identify important predictors and reduce dimensionality.

---

9. Residual Analysis

- **Residuals**: Differences between observed and predicted values.

    - Analysis helps understand if errors are randomly distributed or if patterns exist (indicating model issues).

---

10. Learning Curve

- A plot showing model performance (error) as a function of the training size.
    - Helps diagnose underfitting, overfitting, and whether more data will improve performance.

Variance

High variance

1. High Variance Model on Different Datasets

- High-variance models are very sensitive to training data.
- They fit the training data closely, even capturing noise.
- When trained on different datasets, the model produces widely varying fits.

2. High Variance of Errors on Testing Datasets

- High-variance models struggle to generalize well.
- On different test datasets, the model gives inconsistent predictions.
- This leads to high variance in errors, making the model unreliable.

3. High Variance and High Testing Error

- High-variance models overfit to training data.
- As a result, they have poor generalization and high testing error.

Regularization

Why high weights increase overfitting (using angle/slope analogy)

- A higher slope (like 60°, y = 1.73x) means the model reacts strongly to small input changes.

- A lower slope (like 30°, y = 0.577x) changes the output more gradually.
- The difference in prediction between 60° and 45° is much larger than between 30° and 45°, for the same input shift.
- This shows that **as weights (slope) increase**, the **prediction becomes more unstable**.
- High weights make the model **fit noise**, not just patterns — this is overfitting.
- Lower weights lead to **smoother** and **more general** decision boundaries.

Key Properties of Regularization and Its Effect on Overfitting

- Regularization adds a **penalty for large weights** during training.
- It encourages the model to **keep weight values small**.
- Smaller weights mean **less sensitive** and **more stable** predictions.
- Helps the model **generalize better** to new/unseen data.
- **Reduces overfitting** by preventing the model from memorizing noise.
- Common types: **L2 (Ridge)** and **L1 (Lasso)** regularization.

Classification

⬛

- Binary
- multi-class

⬛Methods to solve classification problem

- Logistic regression: using sigmoid function
- neural networks:

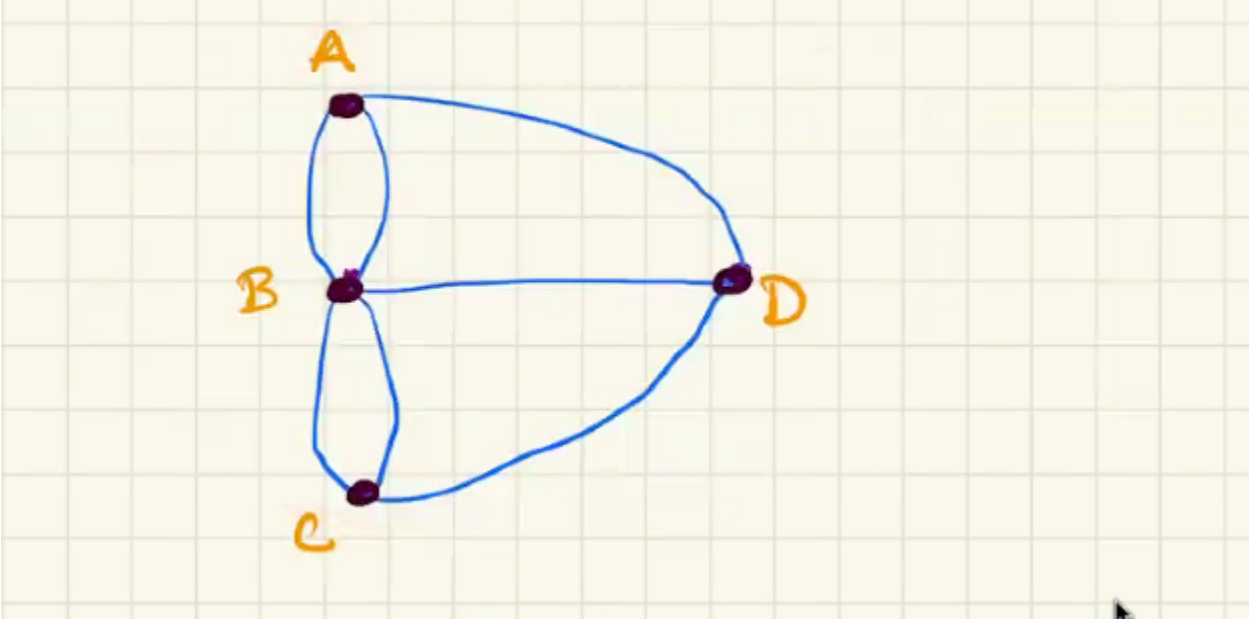
Page rank

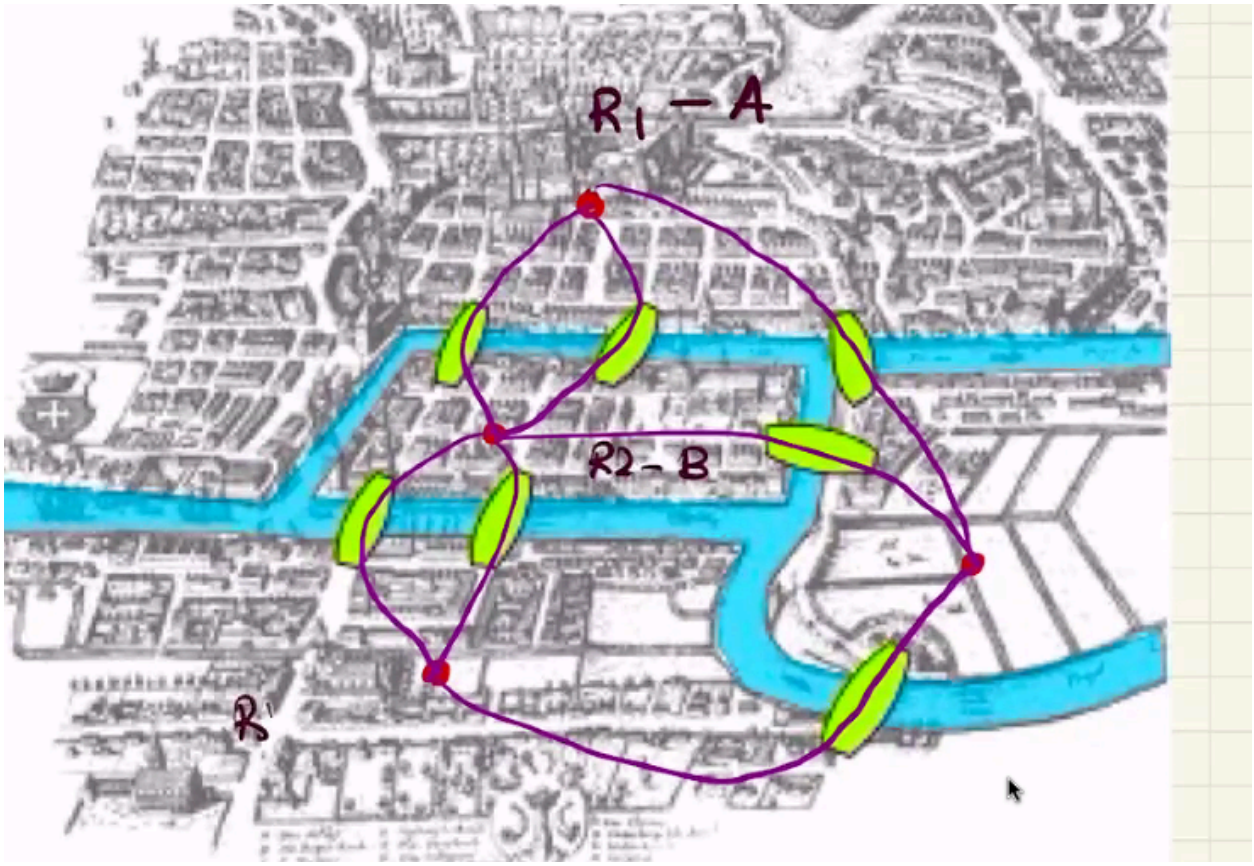
Random Walk: PageRank simulates a random walk where a "web surfer" moves from one page to another based on link structure.

Damping Factor: The algorithm uses a damping factor (typically 0.85) to represent the probability of continuing the random walk or teleporting to a random page.
Link Structure: Pages with more incoming links or higher quality links have higher importance (rank).
Teleportation: The algorithm incorporates teleportation to prevent rank sinks, ensuring every page gets some rank, even if it has no links.

Convlution neural networks

13/12/2024





Graphs

- Nodes, vertex
- Directed, undirected, Degree,
- Adjacency matrix,

Linear transformation

PCA

Eigenvectors

- Represent **directions** in which a transformation acts by **stretching or compressing**.
- Are **non-zero vectors** that **do not change direction** under a linear transformation.
- For a square matrix AAA, an eigenvector vvv satisfies:
  Av=λvA v = \lambda vAv=λv

- Are determined **up to a scalar multiple** — if vvv is an eigenvector, so is 2v2v2v, −v-v−v, etc.
- **Can be orthogonal** if the matrix is symmetric.
- The number of linearly independent eigenvectors determines whether a matrix is **diagonalizable**.

Eigenvalues

- Scalars λ that indicate **how much the eigenvector is stretched or compressed**.
- For an eigenvalue λ\lambdaλ, there exists at least one non-zero vector vvv such that: Av=λvA v = \lambda vAv=λv
- Can be **positive, negative, or complex**, depending on the matrix.
- **The sum of all eigenvalues** equals the **trace** of the matrix.
- **The product of all eigenvalues** equals the **determinant** of the matrix.
- If all eigenvalues are **positive**, the matrix is **positive definite**.

Neural networks

Why Do We Use Deep Learning / Neural Networks?

- Useful for complex problems where traditional ML fails (e.g., image, speech, and text).
- Learns features automatically from raw data — no need for manual feature engineering.
- Can model **non-linear** relationships and very complex patterns.
- Scales well with **large datasets** and performs better as data increases.
- Powers real-world applications like facial recognition, language translation, chatbots, and recommendation systems.

Components of a Neural Network

- **Input Layer**: Takes in the raw data (e.g., pixel values of an image).
- **Hidden Layers**: Middle layers where most of the computation happens.
- **Neurons (Nodes)**: Basic units that compute weighted sums and apply activation functions.
- **Weights & Biases**: Adjustable parameters the network learns during training.
- **Activation Function**: Decides whether a neuron should fire (e.g., ReLU, Sigmoid).
- **Output Layer**: Gives final prediction (e.g., class label, score).

How Neural Networks Work (Simplified)

1. **Input passes forward** through the layers → each neuron computes `weighted sum + bias`, then applies activation.
2. Final output is compared to the actual result using a **loss function**.
3. Using **backpropagation**, the network calculates how wrong it was and adjusts weights to reduce error.
4. This repeats for many rounds (called **epochs**) until the network learns to make good predictions.