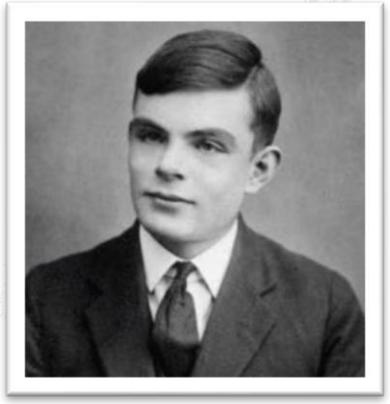
LOVNISH VERMA
Prince Softwares

Introduction to Machine Learning

Foundations and Applications



Lovnish Verma

"We can only see a short distance ahead, but we can see plenty there that needs to be done."— Alan Turing, 1947

Agenda

- What Is Machine Learning?
- How Do We Define Learning?
- How Do We Evaluate Our Networks?
- How Do We Learn Our Network?
- What are datasets and how to handle them?
- Feature sets
- Dataset division: test, train and validation sets, cross validation
- Applications of Machine Learning
- Introduction to Unsupervised Learning and Reinforcement Learning

Figure 1.1: Intro to Machine Learning

What is Machine Learning?

- **Definition**:
  Machine Learning (ML) is a subset of Artificial Intelligence (AI) where systems **learn from data** rather than being explicitly programmed.
- **Arthur Samuel (1959)**:
  "Field of study that gives computers the ability to learn without being explicitly programmed."
- A subfield of AI that focuses on algorithms that *learn patterns from data*
- Learns from **experience (data)** and improves **performance (accuracy, efficiency)** on a **task**
- Shifts from **rule-based programming** ⇨ **data-driven learning**

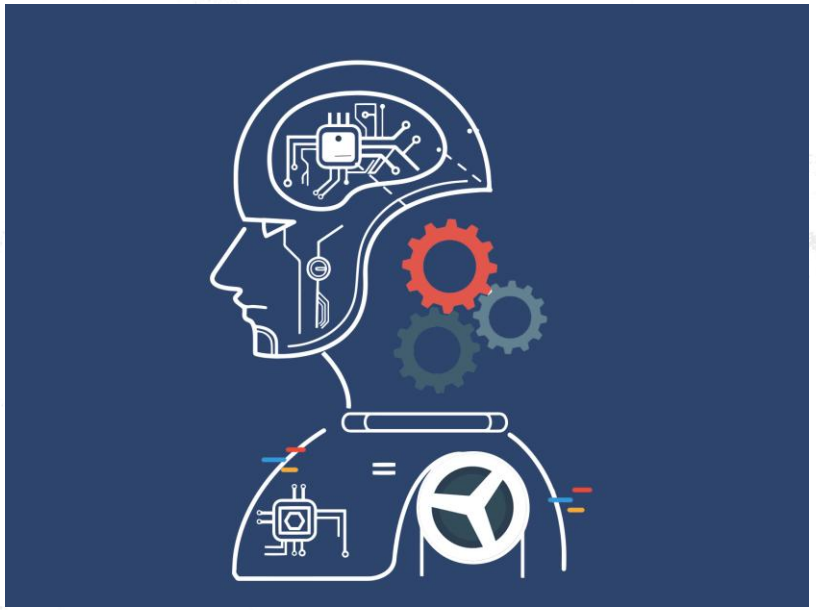


Figure 1.2: Machine Learning Animation

AI vs ML vs Deep Learning

- **Artificial Intelligence (AI):**
  - Broad science of creating machines that mimic human intelligence
  - Includes reasoning, problem-solving, planning, etc.
- **Machine Learning (ML):**
  - Subset of AI where systems learn automatically from data
  - Example: Spam detection, stock prediction
- **Deep Learning (DL):**
  - Subset of ML using **multi-layer neural networks**
  - Excels in vision, speech, and language tasks

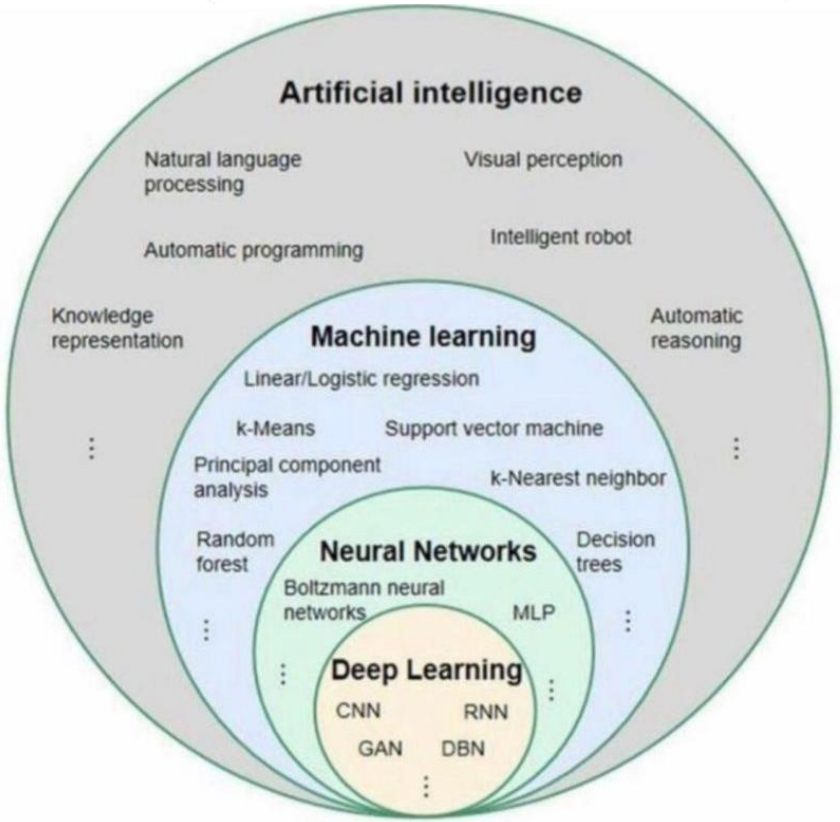


Figure 1.3: Venn Diagram for AI, ML, NLP & DL.

Why Machine Learning is Important?

- Data Explosion: Can process and extract insights from massive, high-dimensional datasets
- Automation: Reduces human effort in repetitive and complex tasks
- Prediction: Enables forecasting and real-time decision-making
- Adaptability: Systems improve with more data (self-learning capability)
- Real-world Applications:
  - **Healthcare:** Disease diagnosis, drug discovery
  - **Finance:** Fraud detection, risk assessment
  - **Retail:** Recommendation systems
  - **Autonomous Systems:** Self-driving cars, robotics
  - **NLP:** Chatbots, translation, speech recognition

Understanding Learning in Machines

LOVNISH VERMA
Prince Softwares

Definition of Learning (Tom Mitchell, 1997)

A computer program is said to learn from experience (E) with respect to some class of tasks (T) and performance measure (P), if its performance at tasks in T, as measured by P, improves with experience E.

- **Experience (E):** Data used for training (examples, past outcomes)
- **Task (T):** The problem to be solved (classification, prediction, etc.)
- **Performance (P):** Metric to measure learning (accuracy, error rate, F1-score)

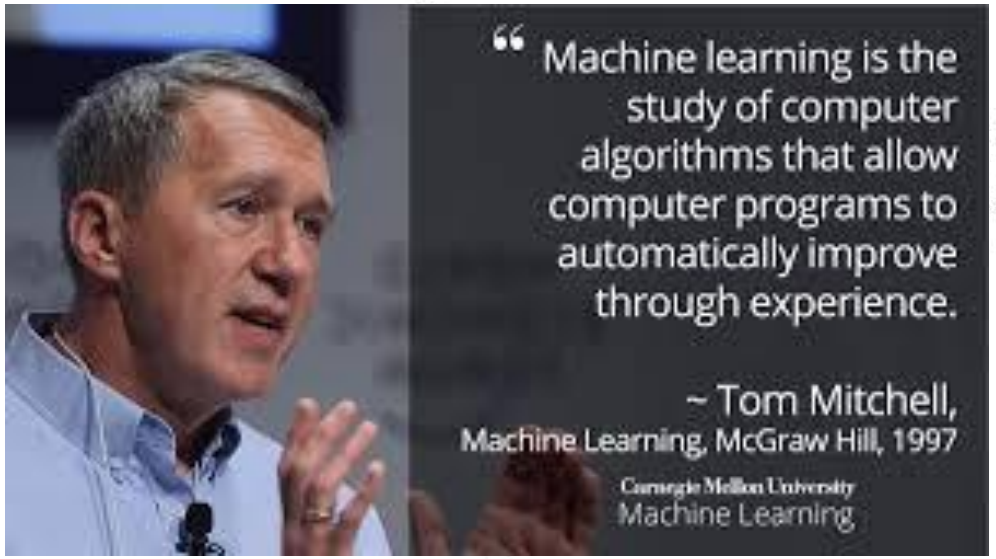


Figure 1.4: Prof. Tom M. Mitchell, Carnegie Mellon University. Author of the textbook “Machine Learning” (1997).

(Example: Spam filter improves (P) in detecting emails (T) as it processes more emails (E))

Types of Tasks in ML

LOVNISH VERMA
Prince Softwares

1. Classification

1. Predict *discrete labels* (e.g., spam vs. not spam, disease present vs. absent)
2. Output: categories/classes (Discrete values (labels).)

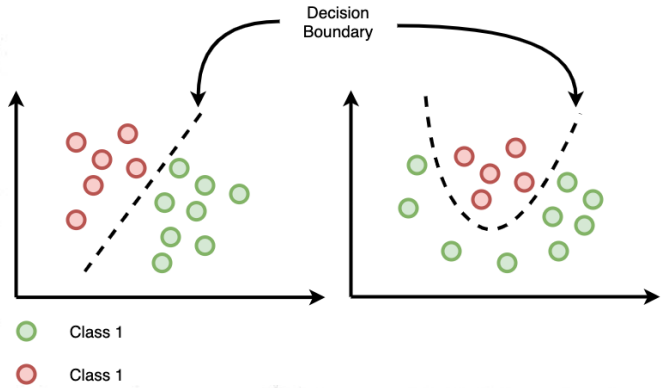


Figure 1.5: Decision boundary for classification

Examples:

- Predicting if a person has **diabetes** → Yes (1) / No (0).
- Predicting if an email is **spam** or **not spam**.
- Predicting the **digit** (0–9) in handwritten images (MNIST dataset).

```
from sklearn.linear_model import LogisticRegression

X = np.array([[0], [1], [2], [3], [4]])   # feature
y = np.array([0, 0, 0, 1, 1])             # target (class labels)

model = LogisticRegression()
model.fit(X, y)

print(model.predict([[1.5], [3.5]]))  # Predict classes
```

Output: `[0 1]`

Types of Tasks in ML

LOVNISH VERMA
Prince Softwares

2. Regression

1. Predict *continuous values*

(e.g., house price, stock market trend)

- Output: Real numbers (e.g., 3.14, 200, 50000).

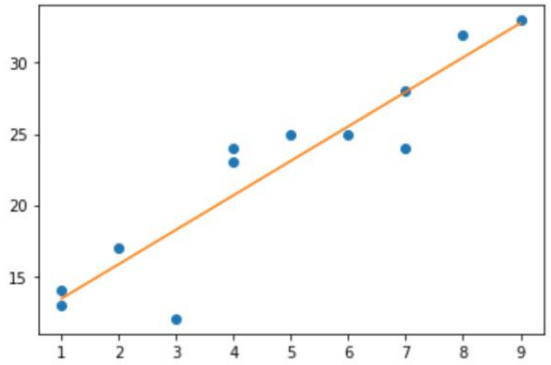


Figure 1.6: Scatter plot with regression line

Examples:

- Predicting **house price** from features like size, location, and rooms.
- Predicting **temperature** tomorrow.
- Predicting **sales revenue** next month.

```python
from sklearn.linear_model import LinearRegression
import numpy as np

X = np.array([[1], [2], [3], [4], [5]])   # feature
y = np.array([2, 4, 6, 8, 10])            # target (continuous)

model = LinearRegression()
model.fit(X, y)

print(model.predict([[6]]))  # Predict value for feature=6
```

Output: `[12.]`

Types of Tasks in ML

LOVNISH VERMA
Prince Softwares

3. Clustering *(Unsupervised task)*

1. Group similar data points without labels
2. Example: customer segmentation, document grouping

Examples:
Customer segmentation in marketing.
Grouping news articles by topic.
Image compression using color clustering.

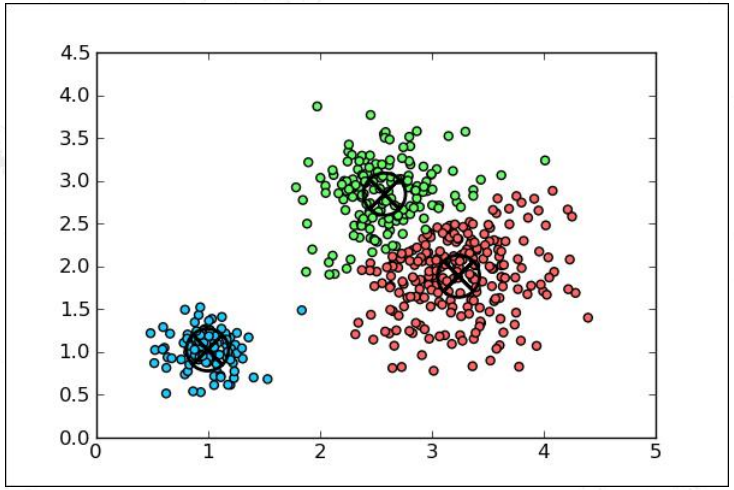


Figure 1.7: Data grouped into clusters using K-Means (colors indicate cluster membership)

```python
from sklearn.cluster import KMeans
import numpy as np

# Sample data (points in 2D space)
X = np.array([
    [1, 2], [1, 4], [1, 0],
    [10, 2], [10, 4], [10, 0]
])

# Create KMeans model with 2 clusters
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X)

# Print the cluster centers
print("Cluster Centers:")
print(kmeans.cluster_centers_)

# Predict cluster for new points
print("Predicted Cluster for [[0,0],[12,3]]:")
print(kmeans.predict([[0, 0], [12, 3]]))

# Labels assigned to training data
print("Labels for training data:")
print(kmeans.labels_)
```

Expected Output (approx):

```
Cluster Centers:
[[ 1.  2.]
 [10.  2.]]
Predicted Cluster for [[0,0],[12,3]]:
[0 1]
Labels for training data:
[0 0 0 1 1 1]
```

Datasets and Data Handling

LOVNISH VERMA
Prince Softwares

Datasets, Features, and Labels

- **Dataset:** Collection of examples used for training/testing ML models
- **Feature (Input X):** Independent variable describing data attributes
- **Label (Output Y):** Dependent variable or target to predict

Example:

Predicting house price ➡ Features: size, bedrooms, location; Label: price

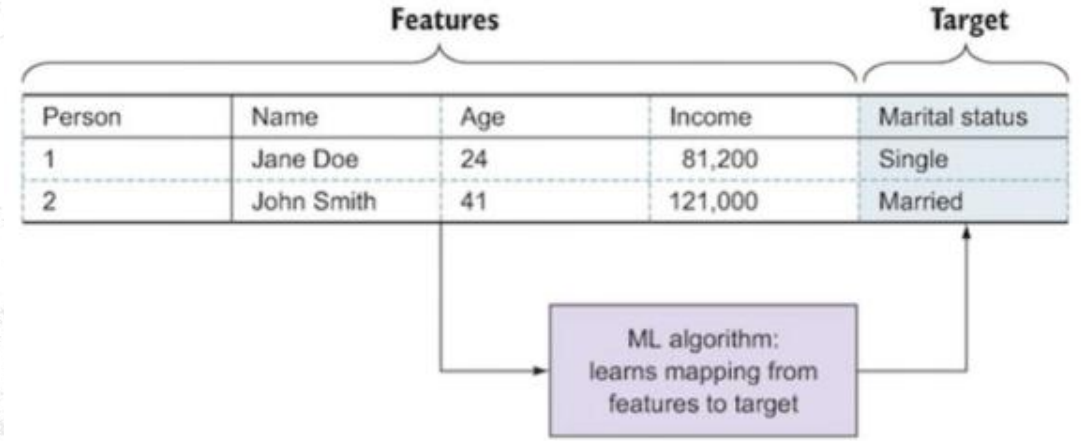


Figure 1.8: Dataset: Features, and Target

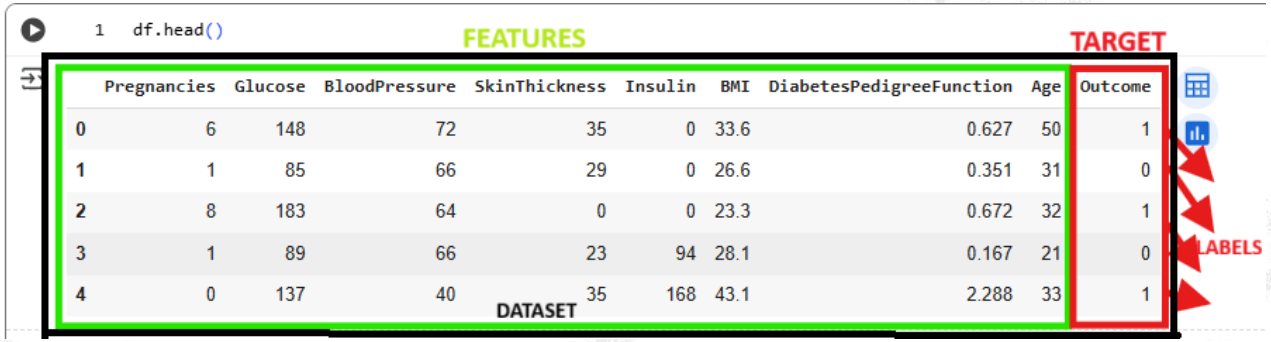


Figure 1.9: Diabetes Datasets, Features, and Target

Train, Test, and Validation Sets

LOVNISH VERMA
Prince Softwares

- **Training Set:** Used to learn model parameters
- **Validation Set:** Used to tune hyperparameters & prevent overfitting
- **Test Set:** Used to evaluate final model performance

(Typical split: 70% train / 15% validation / 15% test)

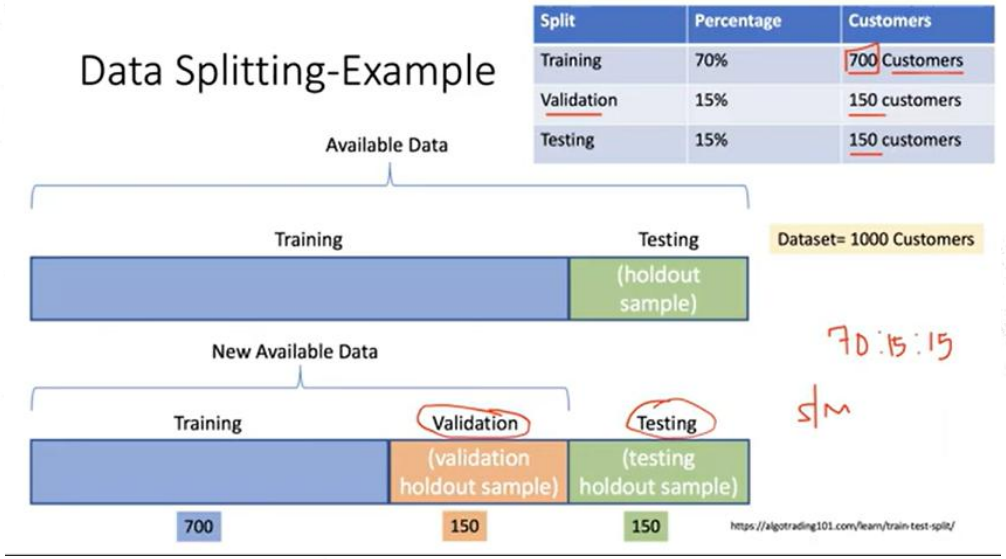


Figure 2.1: Dataset Splitting-Example

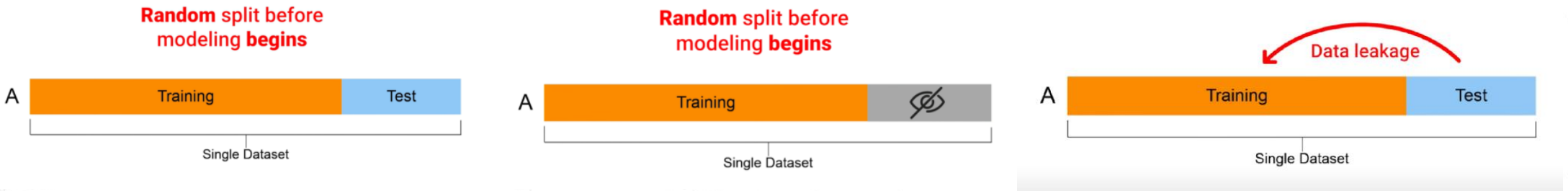


Figure 2.2: Data Random Split, Data Hiding and Data Leakage

Train, Test, and Validation Sets

LOVNISH VERMA
Prince Softwares

Think of it like this:

- **Training Set** ➡ *Textbooks & lecture notes*
  - Model learns patterns from this data.
- **Validation Set** ➡ *Practice exams/quizzes*
  - Used to check understanding, identify weaknesses, and tune hyperparameters.
- **Test Set** ➡ *Final exam*
  - Taken once; provides an unbiased evaluation of true performance on unseen data.

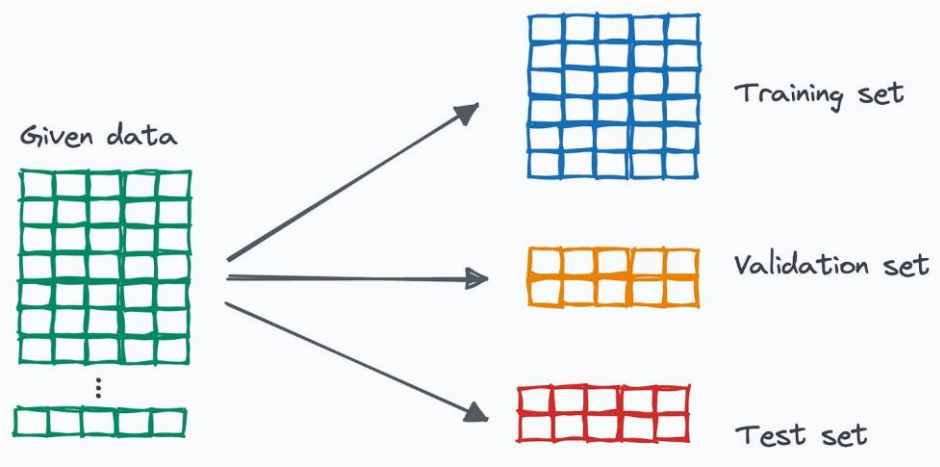


Figure 2.3: Data Splitting-Example

In short: *Train = learn, Validate = refine, Test = evaluate.*

Train, Test, and Validation Sets

LOVNISH VERMA
Prince Softwares

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load data into a pandas DataFrame
df = pd.read_csv("diabetes.csv")

# x = df.drop('Outcome', axis=1)
# y = df['Outcome']

# The line below prepares the data and splits it in one step.
# It separates the features (all columns except 'Outcome') from the target variable ('Outcome').
X_train, X_test, y_train, y_test = train_test_split(df.drop("Outcome", axis=1), df["Outcome"], test_size=0.2)

# Train a logistic regression model on the training data
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = lr_model.predict(X_test)

# You can then evaluate the model's performance
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print(f"Precision: {precision_score(y_test, y_pred)}")
print(f"Recall: {recall_score(y_test, y_pred)}")
print(f"F1 Score: {f1_score(y_test, y_pred)}")
```

```
Accuracy: 0.7727272727272727
Precision: 0.6428571428571429
Recall: 0.574468085106383
F1 Score: 0.6067415730337079
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split

# Assume 'df' is your loaded DataFrame from diabetes.csv
features = df.drop("Outcome", axis=1)
target = df["Outcome"]

# Step 1: Split data into training (80%) and a temporary set (20%)
X_train, X_temp, y_train, y_temp = train_test_split(features, target, test_size=0.2, random_state=42)

# Step 2: Split the temporary set into validation (10%) and testing (10%)
# The new test_size is 0.5 because we are splitting the 20% temporary set in half.
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"Training set shape: {X_train.shape}")
print(f"Validation set shape: {X_val.shape}")
print(f"Testing set shape: {X_test.shape}")

# Now you would:
# 1. Train your model on (X_train, y_train)
# 2. Evaluate and tune it using (X_val, y_val)
# 3. Get the final performance score using (X_test, y_test)
```

```
Training set shape: (614, 8)
Validation set shape: (77, 8)
Testing set shape: (77, 8)
```

Cross-Validation Techniques

LOVNISH VERMA
Prince Softwares

1. k-Fold Cross-Validation

- Divide the dataset into **k equal parts (folds)**
- Train the model **k times**, each time using a different fold as the **test set** and the remaining folds as the **training set**
- Compute performance for each fold and take the **average** → robust estimate of model performance
- Reduces **bias** from a single train-test split

Example:

- Dataset: 1000 emails (spam/not spam)
- 5-Fold CV → each fold has 200 emails
- Model trains on 800 emails, tests on 200; repeat 5 times → average accuracy

Cross-Validation Techniques

LOVNISH VERMA
Prince Softwares

2. Stratified k-Fold Cross-Validation

- Ensures **each fold has the same proportion of classes** as the original dataset
- Important for **imbalanced datasets** (e.g., fraud detection, rare disease prediction)
- Prevents some folds from having too few or no examples of a class

Example:

- Dataset: 1000 credit card transactions (900 normal, 100 fraud)
- Stratified 5-Fold ⇨ each fold has 90 normal + 10 fraud
- Ensures model sees rare cases in all folds

Cross-Validation Techniques

LOVNISH VERMA
Prince Softwares

3. Purpose & Benefits

- **Reliable performance metrics -** avoids misleading results from random splits
- Helps in **hyperparameter tuning** and **model selection**
- Reduces risk of **overfitting** and **underfitting**
- Provides a **better estimate of generalization** on unseen data

Learning Process in Machine Learning

LOVNISH VERMA
Prince Softwares

How Networks Learn

- ML models aim to find a **function** that maps inputs (features) → outputs (labels)
- Learning = **finding model parameters** that minimize prediction error on training data
- Key idea: **adjust model based on feedback (error)** until performance is optimal

Example:

- Email spam classifier
  - Input: Email features (word frequency, sender, etc.)
  - Output: Spam / Not Spam
  - Model updates parameters to reduce misclassification

Evaluating Learning Models

LOVNISH VERMA
Prince Softwares

1. Evaluation Metrics

- **Purpose:** Measure how well a model performs on unseen data.
- **Common Metrics:**
  - **Regression:**
    - Mean Squared Error (MSE), Root Mean Squared Error (RMSE)
    - Mean Absolute Error (MAE)
    - R^2 Score
  - **Classification:**
    - Accuracy, Precision, Recall, F1 Score
    - Confusion Matrix
    - ROC-AUC

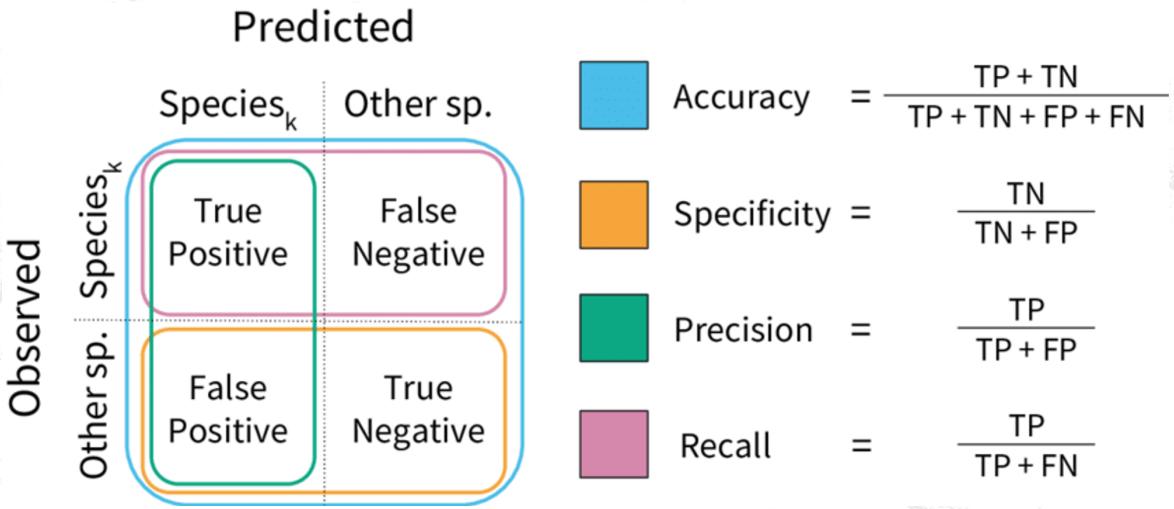


Figure 2.4: Metrics for Classification Model

Regression Metrics

LOVNISH VERMA
Prince Softwares

MAE: *The Mean Absolute Error calculates the average absolute residuals.* It doesn't penalize high errors as much as other evaluation metrics. Every error is treated equally, even the errors of outliers, so this metric is robust to outliers. Moreover, the absolute value of the differences ignores the direction of error.

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

MSE: *The Mean Squared Error calculates the average squared residuals.*

Since the differences between predicted and actual values are squared, It gives more weight to higher errors, so it can be useful when big errors are not desirable, rather than minimizing the overall error.

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

Regression Metrics

LOVNISH VERMA
Prince Softwares

RMSE: *The Root Mean Squared Error calculates the **square root** of the average squared residuals.*

- When you understand MSE, you keep a second to grasp the Root Mean Squared Error, which is just the square root of MSE.
- The good point of RMSE is that it is easier to interpret since the metric is in the scale of the target variable. Except for the shape, it's very similar to MSE: it always gives more weight to higher differences.

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

MAPE: *Mean Absolute Percentage Error calculates the average absolute percentage difference between predicted values and actual values.*

- Like MAE, it disregards the direction of the error and the best possible value is ideally 0.
- For example, if we obtain a MAPE with a value of 0.3 for predicting house prices, it means that, on average, the predictions are below of 30%.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

Regression Metrics

LOVNISH VERMA
Prince Softwares

- **R^2 (R-squared):** R2 score represents the proportion of the variance in the dependent variable that is predictable from the independent variables. An R^2 value close to 1 shows a model that explains most of the variance while a value close to 0 shows that the model does not explain much of the variability in the data. R^2 is used to assess the goodness-of-fit of regression models.
- Formula:

$$R^2 = 1 - \frac{\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}{\sum_{j=1}^{n}(y_j - \bar{y})^2}$$

- Where:
- y_j = Actual value
- $\hat{y}_j$ = Predicted value
- $\bar{y}$ =Mean of the actual values

Classification Metrics

LOVNISH VERMA
Prince Softwares

- **Accuracy:** Accuracy is a fundamental metric used for evaluating the performance of a classification model. It tells us the proportion of correct predictions made by the model out of all predictions.

Accuracy = (True Positives + False Negatives) / Total Cases

- **Precision**: It measures how many of the positive predictions made by the model are actually correct. It's useful when the cost of false positives is high such as in medical diagnoses where predicting a disease when it's not present can have serious consequences. Precision helps ensure that when the model predicts a positive outcome, it's likely to be correct.

Precision = True Positives / (True Positives + False Positives)

- **Recall**: Recall or Sensitivity measures how many of the actual positive cases were correctly identified by the model. It is important when missing a positive case (false negative) is more costly than false positives.

Recall = True Positives / (True Positives + False Negatives)

Classification Metrics

LOVNISH VERMA
Prince Softwares

- **F1 Score**: F1 Score is the harmonic mean of **precision** and **recall**. It is useful when we need a balance between precision and recall as it combines both into a single number. A high F1 score means the model performs well on both metrics. Its range is [0,1].

$$\text{F1-Score} = 2 \times (\text{Precision} \times \text{Recall}) \ / \ (\text{Precision} + \text{Recall})$$

- **Receiver Operating Characteristic (ROC) Curve:** It is a graphical representation of the True Positive Rate (TPR) vs the False Positive Rate (FPR) at different classification thresholds. The curve helps us visualize the trade-offs between sensitivity (TPR) and specificity (1 - FPR) across various thresholds. Area Under Curve (AUC) quantifies the overall ability of the model to distinguish between positive and negative classes.
- **Area Under Curve (AUC)**:It is useful for binary classification tasks. The AUC value represents the probability that the model will rank a randomly chosen positive example higher than a randomly chosen negative example. AUC ranges from 0 to 1 with higher values showing better model performance.

Classification Metrics

- **Confusion Matrix**: For illustration, a confusion matrix of a classification demonstrate may appear that it accurately classified 50 occurrences as positive and 50 occurrences as negative, but erroneously classified 10 occasions as positive and 10 occurrences as negative.

Figure 2.5: Confusion Matrix on Imbalanced Dataset

- **Class Imbalance**: For illustration, on the off chance that a dataset contains 100 positive occurrences and 1000 negative occasions, at that point this dataset is imbalanced.

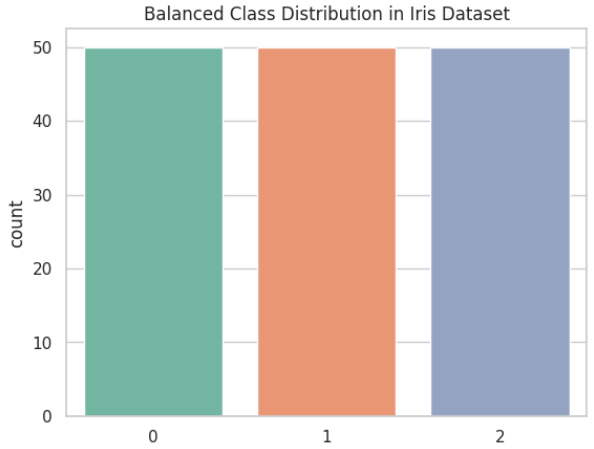


Figure 2.6: Balanced Class Distribution in Iris Dataset

Overfitting and Underfitting: The Core Issues

LOVNISH VERMA
Prince Softwares

- **Overfitting** happens when a model learns too much from the training data, including details that don't matter (like noise or outliers).
- For example, imagine fitting a very complicated curve to a set of points. The curve will go through every point, but it won't represent the actual pattern.
- As a result, the model works great on training data but fails when tested on new data.
- Overfitting models are like students who memorize answers instead of understanding the topic. They do well in practice tests (training) but struggle in real exams (testing).
- **Reasons for Overfitting:**
- High variance and low bias.
- The model is too complex.
- The size of the training data.

Overfitting and Underfitting: The Core Issues

LOVNISH VERMA
Prince Softwares

- **Underfitting** is the opposite of overfitting. It happens when a model is too simple to capture what's going on in the data.
- For example, imagine drawing a straight line to fit points that actually follow a curve. The line misses most of the pattern.
- In this case, the model doesn't work well on either the training or testing data.
- Underfitting models are like students who don't study enough. They don't do well in practice tests or real exams. **Note: The underfitting model has High bias and low variance.**
- **Reasons for Underfitting:**
- The model is too simple, So it may be not capable to represent the complexities in the data.
- The input features which is used to train the model is not the adequate representations of underlying factors influencing the target variable.
- The size of the training dataset used is not enough.
- Excessive regularization are used to prevent the overfitting, which constraint the model to capture the data well.
- Features are not scaled.

Applications of ML in Different Domains

Healthcare

- Disease detection (e.g., cancer, diabetes, heart disease)
- Drug discovery & development
- Patient risk prediction
- Predictive diagnostics & personalized treatment

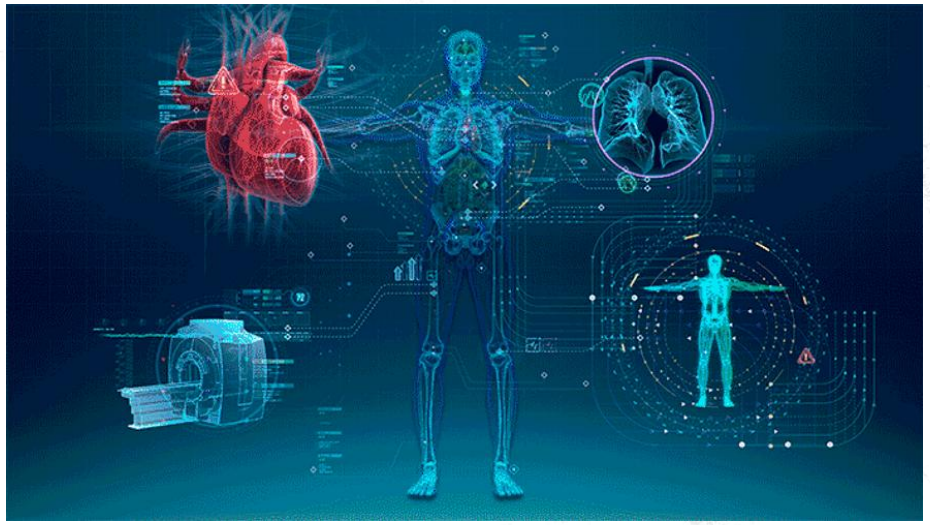


Figure 2.7: Applications of ML in Healthcare

Applications of ML in Different Domains

LOVNISH VERMA
Prince Softwares

Finance

- Credit scoring
- Fraud detection
- Algorithmic trading, risk assessment
- AI chatbots & virtual assistants

Figure 2.8: Applications of ML in Finance

Applications of ML in Different Domains

Agriculture

- Smart irrigation & resource management
- Crop yield prediction & disease monitoring
- Precision farming using sensors & drones

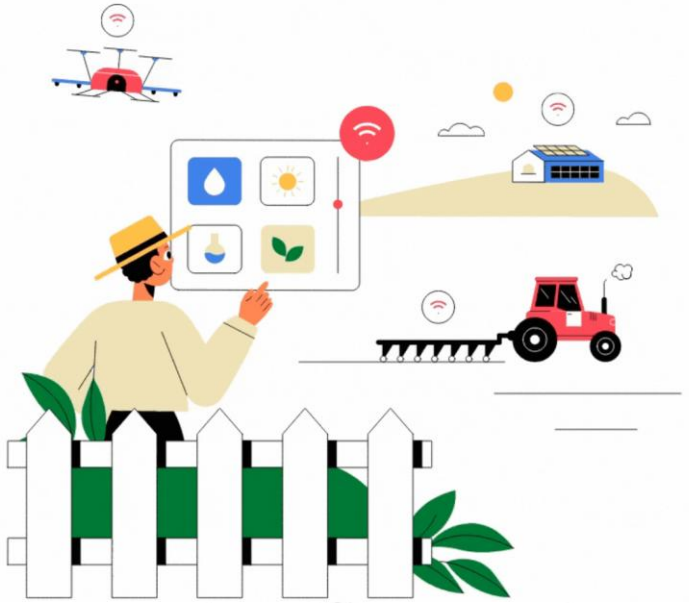


Figure 2.9: Applications of ML in Agriculture

Applications of ML in Different Domains

Education

- Personalized learning platforms
- AI-powered tutors & grading systems
- Intelligent content recommendations

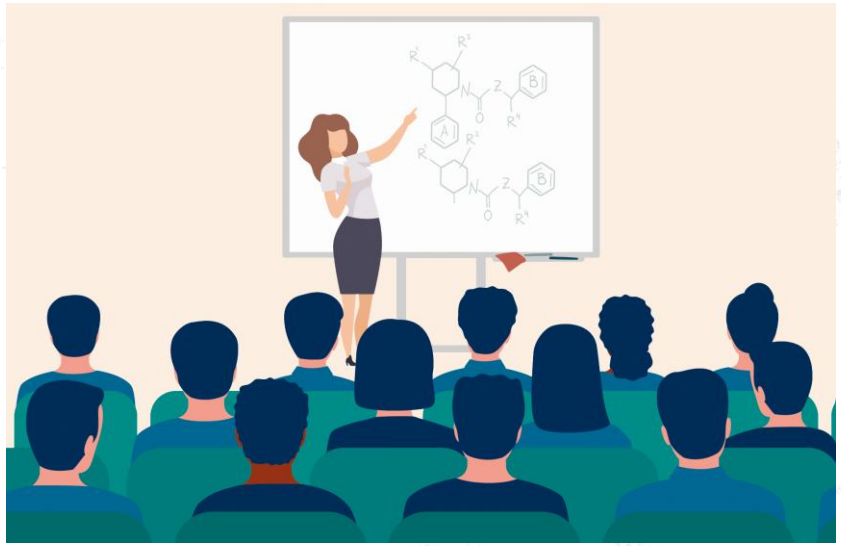


Figure 3.1: Applications of ML in Education

Applications of ML in Different Domains

Autonomous Systems

- Self-driving cars, drones & autonomous vehicles
- Traffic management & route optimization
- Robotics
- Predictive maintenance of vehicles

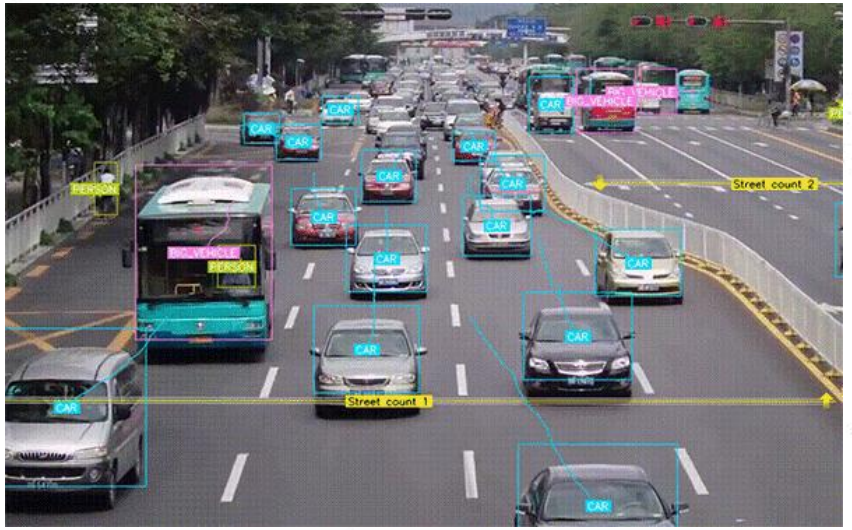


Figure 3.2: Applications of ML in Autonomous Systems

Applications of ML in Different Domains

Natural Language Processing (NLP)

- Sentiment analysis
- Chatbots
- machine translation
- speech recognition

Figure 3.3: Applications of ML in NLP

Machine Learning Paradigms

1. Unsupervised Learning

- **Definition:**
  Machine learning paradigm where the model learns patterns, structures, or relationships from **unlabeled data**. There are no predefined outputs or targets.

Objective:
Discover hidden structures, group similar data points, reduce dimensionality, or detect anomalies.

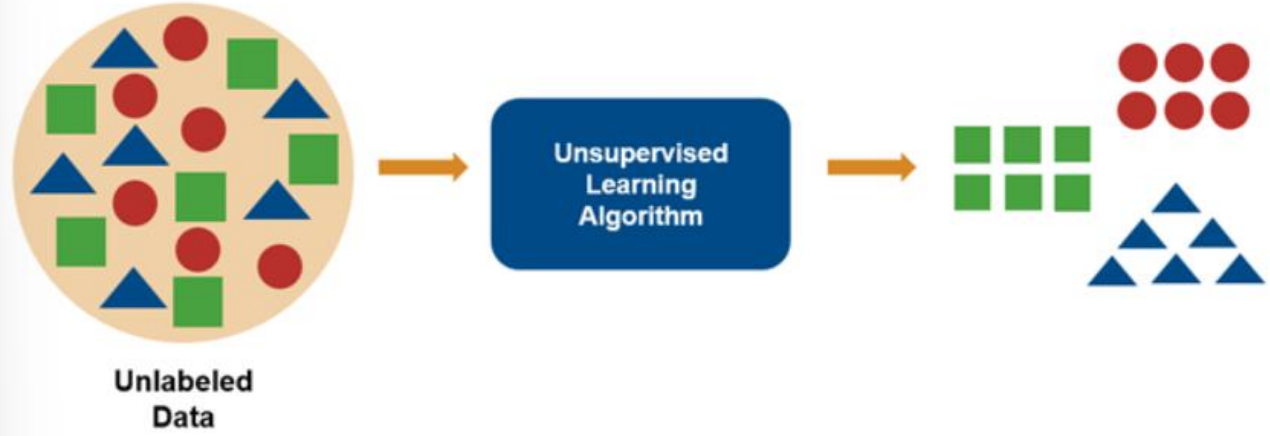


Figure 3.3: Organizing unlabeled data into groups using unsupervised learning.

Machine Learning Paradigms (Unsupervised Learning)

- **Key Techniques:**
  - **Clustering:** K-Means, DBSCAN – Grouping similar data points
  - **Dimensionality Reduction:** PCA, t-SNE – Simplifying data while preserving structure
  - **Anomaly Detection:** Identifying outliers or rare events
- **Applications:**
  - Customer segmentation in marketing
  - Gene expression pattern discovery
  - Fraud detection in banking

Machine Learning Paradigms

Reinforcement Learning (RL)

- **Definition:**
  A learning paradigm in which an **agent interacts with an environment**, taking actions to **maximize cumulative reward** based on feedback.
  The agent learns optimal behavior over time.

Objective:

Learn a policy mapping states to actions to achieve the best long-term reward.

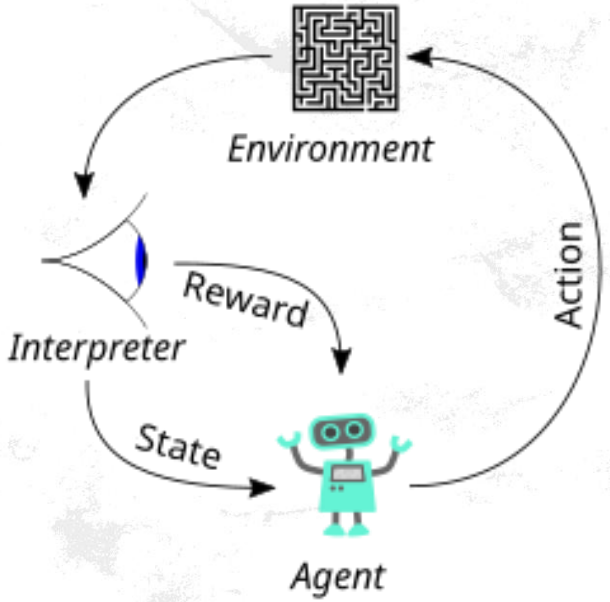


Figure 3.4: Agent-Environment Interaction

Machine Learning Paradigms(Reinforcement Learning)

LOVNISH VERMA
Prince Softwares

- **Key Concepts:**
  - **Agent:** The learner or decision-maker
  - **Environment:** The world the agent interacts with
  - **Action:** Choices made by the agent
  - **Reward:** Feedback signal for performance
  - **Policy:** Strategy that defines agent's actions in each state
- **Applications:**
  - Game AI: AlphaGo, Chess and video game agents
  - Robotics: Autonomous navigation and task learning
  - Autonomous vehicles: Self-driving cars, drones
  - Recommendation systems: Optimizing suggestions via trial-and-error

Thank You

LOVNISH VERMA
Prince Softwares

https://lovnishverma.github.io/

@lovnishverma

@lovnishverma