Buildables Fellowship 2025

ML/DL Track

Team lead: Abdullah Naeem

Week 1 - Intro to ML Algos

Assignment #01

Table of Contents

1 Objectives 4
Machine Learning Models 5
Machine Learning 5
Types of Machine Learning Models 5
Supervised Learning 5
Unsupervised Learning 5
Reinforcement Learning 5
Importance of Model Selection and Evaluation 6
Bayesian Belief Networks (BBN) 6
Probabilistic Graphical Models 6
Conditional Independence 6
Structure of a BBN 6
Nodes (Variables) 6
Directed Edges (Causal Relationships) 7
Conditional Probability Distributions (CPDs) 7
Simple Bayesian Belief Network (BBN) Code with Explanation 7
Libraries: pgmpy, pyAgrum 7
Code Explanation 7
K-Nearest Neighbors (KNN) Algorithm 8
Instance-Based Learning 9
Non-Parametric Model 9
How KNN Works 9
Choosing the Value of K 9
Implementation of KNN 9
Libraries to Import 9
Code Example 10
Explanation 10
Applications of KNN 10
Recommendation Systems 10
Image Recognition 10
Fraud Detection 10
Support Vector Machines (SVM) 11

Classification vs. Regression....11
Hyperplanes and Support Vectors....11
Mathematical Intuition of SVM....11
Maximal Margin Classifier....11
Soft Margin vs. Hard Margin....11
Kernel Trick (Linear, Polynomial, RBF, Sigmoid)....11
Implementation of SVM....12
Libraries to Import....12
Code Example....12
Random Forest....12
How Random Forest Works....12
Bootstrapping (Bagging)....12
Feature Selection in Decision Trees....12
Majority Voting (Classification) / Averaging (Regression)....13
Implementation of Random Forest....13
Libraries to Import....13
Code Example....13
Model Evaluation in Machine Learning....13
Understanding Model Performance....14
Checking Model Accuracy....14
Avoiding Overfitting and Underfitting....14
Checking for Overfitting....14
Comparing Different Models....15
Comparing Accuracy of Two Models....15
Evaluation Metrics....15
Classification Metrics....15
Precision....16
Recall (Sensitivity)....16
F1 Score....16
Regression Metrics....17
Mean Squared Error (MSE)....17
Evaluation Methods....18
Confusion Matrix....18
Key Terms....18

Code Example....18
ROC Curve (Receiver Operating Characteristic)....19
True Positive Rate (TPR) (Sensitivity or Recall)....19
False Positive Rate (FPR)....19
AUC Score (Area Under the Curve)....19
Lab Tasks....20
1. Support Vector Machines (SVM) – Decision Boundaries....20
2. Random Forest – Overfitting vs. Generalization....20
3. Confusion Matrix and ROC Curve – Model Evaluation....20
4. K-Nearest Neighbors (KNN) – Choosing Optimal K....20
5. Bayesian Belief Networks (BBN) – Probabilistic Decision Making....20
Bonus Challenge....21

1 Objectives

- Understand the fundamental concepts of machine learning, model evaluation, and data preprocessing by working with real-world datasets.
- Explain the importance and applications of feature selection, classification, and regression in domains like finance, healthcare, and fraud detection.
- Identify and apply different data preprocessing techniques, including handling missing values, standardization, encoding categorical variables, and feature scaling for models like SVM and KNN.
- Implement and analyze classification models such as Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Random Forest, understanding their strengths and limitations in different scenarios.
- Perform Bayesian Belief Network (BBN) analysis to model probabilistic reasoning and decision-making under uncertainty.
- Evaluate machine learning models using Confusion Matrix, ROC Curve, and key metrics (Accuracy, Precision, Recall, F1-score) to interpret model performance.
- Optimize hyperparameters in KNN and Random Forest models to improve classification accuracy and prevent overfitting.
- Use visualization techniques such as heatmaps, scatter plots, confusion matrices, and ROC curves to interpret classification performance.
- Automate machine learning pipelines for preprocessing, training, and evaluation using Python libraries such as scikit-learn, pandas, seaborn, and matplotlib.
- Compare different machine learning models (SVM, KNN, Random Forest, Bayesian Networks) based on evaluation metrics to select the best-performing algorithm.

- Recognize different types of classification and regression problems and choose the most suitable model for real-world applications like loan approval, fraud detection, and medical diagnostics.

- Improve model performance through hyperparameter tuning and ensemble learning techniques, such as Random Forest Bagging and SVM kernel selection.

Machine Learning Models

Machine Learning (ML) is a branch of artificial intelligence (AI) that enables systems to learn and improve from experience without being explicitly programmed. ML models analyze patterns in data to make predictions or decisions. These models are widely used in various industries, including finance, healthcare, and marketing. For example, an ML model in healthcare can predict patient disease risks based on historical medical records.

Machine Learning

Machine Learning is the process of training a computer system to recognize patterns and make decisions based on data without human intervention. It uses statistical techniques and algorithms to learn from past experiences and improve over time. A common example is email spam filtering, where an ML model classifies emails as spam or non-spam based on past data.

Types of Machine Learning Models

ML models can be broadly classified into three types: **Supervised Learning, Unsupervised Learning, and Reinforcement Learning**. Each type is suited for different kinds of problems. For instance, Supervised Learning is used for predicting house prices based on historical data, Unsupervised Learning is used for customer segmentation in marketing, and Reinforcement Learning is used for training self-driving cars.

Supervised Learning

Supervised Learning involves training a model using labeled data, where input-output pairs are provided. The model learns the mapping function and predicts outputs for unseen data. Common supervised learning algorithms include **Linear Regression, Decision Trees, and Support Vector Machines (SVM)**. For example, in **loan approval systems**, a supervised learning model predicts whether a loan should be approved based on historical applicant data.

Unsupervised Learning

Unsupervised Learning deals with datasets that do not have labeled outputs. The algorithm identifies hidden patterns and structures within the data. Common techniques include **Clustering (e.g., K-Means, DBSCAN)** and **Dimensionality Reduction (e.g., PCA)**. A real-world example is **market basket analysis**, where stores analyze purchase behavior to group similar products together for better recommendations.

Reinforcement Learning

Reinforcement Learning (RL) is a type of ML where an agent learns by interacting with its environment and receiving rewards or penalties. The goal is to maximize cumulative rewards over time. RL is commonly used in **robotics, game playing (AlphaGo), and self-driving cars**. A simple example is **a chess-playing AI** that improves its strategy by continuously playing

games and learning from past moves.

Importance of Model Selection and Evaluation

Choosing the right model is crucial for achieving high accuracy and efficiency in machine learning tasks. Different problems require different models based on factors like data size, complexity, and interpretability. Additionally, evaluating the model's performance ensures that it generalizes well to unseen data. Common evaluation metrics include **accuracy, precision, recall, F1-score, and AUC-ROC**. For example, in a **fraud detection system**, high precision is more important than overall accuracy to minimize false positives.

Bayesian Belief Networks (BBN)

A **Bayesian Belief Network (BBN)** is a **probabilistic graphical model** that represents a set of variables and their probabilistic dependencies using **Bayes' theorem**. It is used to model complex systems where uncertainty is present, making it ideal for reasoning under uncertainty in AI applications. A real-world example of a **BBN** is **medical diagnosis**, where symptoms (observed variables) are connected to potential diseases (hidden variables) to infer probabilities of illness.

Probabilistic Graphical Models

A **BBN** is a type of **probabilistic graphical model (PGM)** that uses **graph theory** to represent relationships between variables. It is composed of **nodes (variables)** and **directed edges (dependencies)**. These models are used in fields like **finance, healthcare, and robotics** to predict outcomes based on uncertain data. For example, a **financial risk assessment model** can predict market trends based on historical data and economic indicators.

Conditional Independence

Conditional independence is a fundamental concept in BBNs, where two variables are independent given the knowledge of a third variable. This allows for **simplification of probability calculations**. For example, in **disease diagnosis**, knowing that a patient has a fever and a cough may indicate flu, making other unrelated symptoms irrelevant to the diagnosis.

Structure of a BBN

A **BBN** is structured as a **Directed Acyclic Graph (DAG)**, consisting of:

Nodes (Variables)

- Each node represents a **random variable** (e.g., "Fever", "Flu", "Cough").
- Nodes can be **observable (evidence)** or **hidden (latent variables)**.

Directed Edges (Causal Relationships)

- A directed edge represents a **causal dependency** between two nodes.
- Example: **"Flu" → "Fever"** (Having flu increases the probability of having a fever).

Conditional Probability Distributions (CPDs)

- Each node has a **Conditional Probability Table (CPT)** representing the probability of each possible state given the parent nodes.
- Example: Given a flu diagnosis, the probability of fever might be **0.9**, while without flu, the probability of fever might be **0.2**.

Simple Bayesian Belief Network (BBN) Code with Explanation

To make the **Bayesian Belief Network (BBN)** easier to understand, let's implement a simple example where we model a **student's grade** based on their **difficulty level of the subject** and whether they **studied or not**.

Libraries: pgmpy, pyAgrum

- pgmpy is a Python library for **probabilistic graphical models**.
- pyAgrum is another library for **Bayesian networks and decision analysis**.

Code Explanation

We will create a **Bayesian Network** with the following dependencies:

- A student's **grade** depends on:
  - **Whether they studied or not** (Studied → Grade)
  - **How difficult the subject is** (Difficulty → Grade)
- The **difficulty of the subject** and whether the student **studied** are independent variables.

Define the Bayesian Network Structure

- Create a **BayesianModel** with directed edges (Studied → Grade, Difficulty → Grade).
- Represents **causal relationships** between variables.

Define Conditional Probability Distributions (CPDs)

- Studied:
  - 60% students do not study, 40% do.
- Difficulty:
  - 70% subjects are easy, 30% are hard.
- Grade:
  - Depends on both Studied and Difficulty.
  - Probability table is assigned based on different cases.

Add CPDs to the Model

- Attach all probability tables to the Bayesian model.
- assert model.check_model() ensures validity.

Perform Inference using Variable Elimination

- Use VariableElimination(model) for probability calculations.

Query Example

- Compute the probability of different grades (A, B, C) **given that the student studied**.

Expected Output

- The model returns **probabilities of getting A, B, or C** when Studied = Yes.

Interpretation

- If a student **studies**, they have:
  - **20% chance** of getting an **A**.
  - **40% chance** of getting a **B**.
  - **40% chance** of getting a **C**.

This structured explanation breaks down each step in a **simple and clear** way

```python
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination

# Step 1: Define the Bayesian Network Structure
model = BayesianModel([('Studied', 'Grade'), ('Difficulty', 'Grade')])

# Step 2: Define Conditional Probability Distributions (CPDs)
cpd_studied = TabularCPD(variable='Studied', variable_card=2, values=[[0.6], [0.4]])  # 60% students do not study, 40% do

cpd_difficulty = TabularCPD(variable='Difficulty', variable_card=2, values=[[0.7], [0.3]])  # 70% easy, 30% hard

cpd_grade = TabularCPD(
    variable='Grade', variable_card=3,  # Grade has 3 states: A, B, C
    values=[
        # Given Studied = No & Difficulty = Easy
        [0.3, 0.05, 0.2, 0.02],  # Probability of Grade = A
        [0.5, 0.25, 0.4, 0.2],   # Probability of Grade = B
        [0.2, 0.7, 0.4, 0.78]    # Probability of Grade = C
    ],
    evidence=['Studied', 'Difficulty'], evidence_card=[2, 2]
)

# Step 3: Add CPDs to the Model
model.add_cpds(cpd_studied, cpd_difficulty, cpd_grade)

# Step 4: Verify the Model
assert model.check_model()

# Step 5: Perform Inference
inference = VariableElimination(model)

# Query: What is the probability of getting an 'A' given that the student studied?
result = inference.query(variables=['Grade'], evidence={'Studied': 1})

print(result)
```

K-Nearest Neighbors (KNN) Algorithm

K-Nearest Neighbors (**KNN**) is a **non-parametric, instance-based learning algorithm** used for **classification and regression**. It classifies a new data point based on the majority class of its nearest neighbors.

- Example: If we want to classify a **new fruit** as either an **apple or an orange**, KNN will look at the closest known fruits and assign the new fruit to the majority class.

Instance-Based Learning

KNN is an **instance-based learning algorithm**, meaning it **does not learn a model** beforehand but **stores the entire dataset** and makes predictions when queried.

- **Example**: A doctor diagnosing a patient based on similar past cases.

Non-Parametric Model

KNN is a **non-parametric algorithm**, meaning it **does not assume any specific distribution** for the data. It makes decisions based purely on **proximity (distance) to training examples**.

- **Example**: It can classify both **linear and non-linear data** without prior assumptions.

How KNN Works

- Store all training data points.
- Choose the number of neighbors (K).
- Calculate the distance between the query point and all training data points.
- Select the K nearest points.
- Assign the most common label (classification) or average (regression).

Choosing the Value of K

- **Small K**: More sensitive to noise, can lead to overfitting.
- **Large K**: Smoother decision boundary, reduces noise but can underfit.

Common approach: Use **cross-validation** to find the best K.

Implementation of KNN

Libraries to Import

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
```

Code Example

```
# Load Dataset (Iris dataset - Classification)
iris = load_iris()
X, y = iris.data, iris.target

# Split dataset into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create KNN model
knn = KNeighborsClassifier(n_neighbors=3)  # Choosing K=3

# Train the model
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

Explanation

- **Load dataset**: We use the Iris dataset.
- **Split data**: 80% training, 20% testing.
- **Train KNN model**: We use K=3 neighbors.
- **Predict**: Classifies test data.
- **Evaluate**: Computes accuracy.

Applications of KNN

Recommendation Systems

Netflix/Amazon uses KNN to suggest movies/products based on user preferences.

Image Recognition

KNN classifies handwritten digits in **OCR (Optical Character Recognition)** systems.

Fraud Detection

Banks use KNN to detect unusual **transaction patterns** to flag fraud.

Support Vector Machines (SVM)

Support Vector Machine (**SVM**) is a **supervised learning algorithm** used for **classification and regression** tasks. It works by finding the best decision boundary (hyperplane) that separates different classes.

- **Example:** In a spam filter, SVM can classify emails as **spam or not spam** by finding the best boundary between the two categories.

Classification vs. Regression

- **SVM for Classification:** Finds the **optimal hyperplane** to separate data into different classes.
- **SVM for Regression:** Uses a margin of tolerance (epsilon) instead of a strict boundary to make predictions.
- **Example:** Predicting **house prices** using SVM regression

Hyperplanes and Support Vectors

- A **hyperplane** is a decision boundary that **separates different classes**.
- **Support vectors** are the data points closest to the hyperplane, influencing the decision boundary.
- **Example:** If we classify dogs vs. cats, the closest dog and cat points to the boundary are **support vectors**.

Mathematical Intuition of SVM

Maximal Margin Classifier

- SVM tries to maximize the **margin** (distance) between the closest data points of both classes and the hyperplane.

Soft Margin vs. Hard Margin

- **Hard Margin:** Used when data is **perfectly separable**.
- **Soft Margin:** Allows **some misclassification** for better generalization.

Kernel Trick (Linear, Polynomial, RBF, Sigmoid)

SVM can handle **non-linear data** using different **kernels**:

- **Linear Kernel:** Used when data is **linearly separable**.
- **Polynomial Kernel:** Works well when data has **curved decision boundaries**.
- **Radial Basis Function (RBF) Kernel:** Captures **complex patterns**.
- **Sigmoid Kernel:** Works in **neural network-like models**.

Implementation of SVM

Libraries to Import

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
```

Code Example

```
# Split Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train SVM Model
svm = SVC(kernel='linear')  # Using a linear kernel
svm.fit(X_train, y_train)

# Make Predictions
y_pred = svm.predict(X_test)

# Evaluate Model
accuracy = accuracy_score(y_test, y_pred)
```

Random Forest

Random Forest is an **ensemble learning technique** that **combines multiple decision trees** to improve accuracy and prevent overfitting.

- **Example:** Predicting stock prices using multiple decision trees.

How Random Forest Works

Bootstrapping (Bagging)

- Creates multiple subsets of data and trains a **decision tree on each subset**.

Feature Selection in Decision Trees

- Randomly selects a subset of features for each tree to **reduce overfitting**.

Majority Voting (Classification) / Averaging (Regression)

- **Classification:** Majority voting decides the final class.
- **Regression:** The final prediction is the **average** of all trees.

Implementation of Random Forest

Libraries to Import

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
```

Code Example

```
# Split Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Random Forest Model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Make Predictions
y_pred = rf.predict(X_test)

# Evaluate Model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

Model Evaluation in Machine Learning

Model evaluation is crucial to determine how well a machine learning model performs on unseen data. It helps assess the model's **accuracy, robustness, and generalizability** before deploying it in real-world applications. Without proper evaluation, a model may provide misleading results or fail in production environments.
For example, consider a **spam detection system** that classifies emails as spam or not spam. If we evaluate the model solely based on accuracy, we may miss important insights like how well it identifies actual spam emails without wrongly flagging legitimate ones.

Understanding Model Performance

Model performance is assessed using various metrics based on the type of problem (classification or regression). A well-performing model should **balance bias and variance**, ensuring it generalizes well to new data.

Checking Model Accuracy

```
from sklearn.metrics import accuracy_score
y_actual = [1, 0, 1, 1, 0, 0, 1, 0]
y_pred = [1, 0, 1, 0, 0, 1, 1, 0]
accuracy = accuracy_score(y_actual, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")  # Output: 0.75 (75%)
```

Avoiding Overfitting and Underfitting

- **Overfitting** occurs when the model learns too much from the training data, capturing noise rather than general patterns. It performs well on training data but poorly on test data.
- **Underfitting** happens when the model is too simple and fails to capture important patterns in the data.

To prevent overfitting, **techniques like cross-validation, pruning (for decision trees), and regularization (L1/L2 in regression models)** can be used.

Checking for Overfitting

```
train_accuracy = 0.98  # 98% on training data
test_accuracy = 0.75   # 75% on test data

if train_accuracy - test_accuracy > 0.1:
    print("The model might be overfitting.")
else:
    print("The model is generalizing well.")
```

Comparing Different Models

When building machine learning models, it is essential to compare multiple models and select the best-performing one based on evaluation metrics. **No single model is best for all tasks**. For example, in a **fraud detection system**, **Random Forest may perform better** than Logistic Regression because it captures complex relationships. However, **if interpretability is required, Logistic Regression is preferred**.

Comparing Accuracy of Two Models

```python
from sklearn.metrics import accuracy_score

# Assume predictions from two models
y_pred_model1 = [1, 0, 1, 1, 0, 1, 1, 0]  # Model 1
y_pred_model2 = [1, 0, 1, 0, 0, 1, 0, 0]  # Model 2

accuracy_model1 = accuracy_score(y_actual, y_pred_model1)
accuracy_model2 = accuracy_score(y_actual, y_pred_model2)

print(f"Model 1 Accuracy: {accuracy_model1:.2f}")
print(f"Model 2 Accuracy: {accuracy_model2:.2f}")

# Choosing the better model
best_model = "Model 1" if accuracy_model1 > accuracy_model2 else "Model 2"
print(f"The better model is: {best_model}")
```

Evaluation Metrics

Classification Metrics

Accuracy – Overall Correct Predictions
Accuracy measures the percentage of correctly predicted instances out of total predictions.

```python
from sklearn.metrics import accuracy_score
y_actual = [1, 0, 1, 1, 0, 0, 1, 0]
y_pred = [1, 0, 1, 0, 0, 1, 1, 0]

accuracy = accuracy_score(y_actual, y_pred)
print(f"Accuracy: {accuracy:.2f}")  # Output: Accuracy: 0.75
```

Precision

How Many Predicted Positives are Correct?
Precision measures the proportion of true positive predictions out of all positive predictions.

```python
from sklearn.metrics import precision_score
precision = precision_score(y_actual, y_pred)
print(f"Precision: {precision:.2f}")
```

Recall (Sensitivity)

How Many Actual Positives are Identified?
Recall (also called Sensitivity) measures how many actual positive instances were correctly predicted.

```python
from sklearn.metrics import recall_score
recall = recall_score(y_actual, y_pred)
print(f"Recall: {recall:.2f}")
```

F1 Score

Harmonic Mean of Precision and Recall
The F1 score balances **Precision and Recall** by calculating their harmonic mean.

```python
from sklearn.metrics import f1_score
f1 = f1_score(y_actual, y_pred)
print(f"F1 Score: {f1:.2f}")
```

Regression Metrics

Mean Squared Error (MSE)

MSE calculates the average squared differences between actual and predicted values. It **penalizes large errors** more.

```
from sklearn.metrics import mean_squared_error
y_actual = [3.0, -0.5, 2.0, 7.0]
y_pred = [2.5, 0.0, 2.1, 7.8]

mse = mean_squared_error(y_actual, y_pred)
print(f"MSE: {mse:.2f}")
```

Root Mean Squared Error (RMSE)

RMSE is the **square root of MSE**, making it more interpretable in the original unit.

```
import numpy as np
rmse = np.sqrt(mse)
print(f"RMSE: {rmse:.2f}")
```

Mean Absolute Error (MAE)

MAE measures the average absolute difference between actual and predicted values.

```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_actual, y_pred)
print(f"MAE: {mae:.2f}")
```

R-Squared (R^2 Score)

R^2 Score measures how well the regression model explains variance in the data. A score of **1.0 indicates perfect prediction**.

```
from sklearn.metrics import r2_score
r2 = r2_score(y_actual, y_pred)
print(f"R² Score: {r2:.2f}")
```

Evaluation Methods

Confusion Matrix

A **Confusion Matrix** is a table that helps evaluate the performance of a classification model by showing the counts of **true and false predictions** for each class. It provides insight into how well a model is distinguishing between different classes.
For example, in **a spam detection system**, a confusion matrix can show how many emails were **correctly classified as spam or non-spam** and how many were misclassified.

Key Terms

- **True Positive (TP)** – Correctly predicted **positive** instances.
- **True Negative (TN)** – Correctly predicted **negative** instances.
- **False Positive (FP) (Type I Error)** – Incorrectly predicted as positive when it was actually negative.
- **False Negative (FN) (Type II Error)** – Incorrectly predicted as negative when it was actually positive.

Actual ↓ / Predicted →	**Positive (Spam)**	**Negative (Not Spam)**
Positive (Spam)	**TP** (Correct)	**FN** (Missed)
Negative (Not Spam)	**FP** (Wrongly Marked as Spam)	**TN** (Correct)

Code Example

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Actual and predicted values (1 = spam, 0 = not spam)
y_actual = [1, 0, 1, 1, 0, 0, 1, 0]
y_pred = [1, 0, 1, 0, 0, 1, 1, 0]

# Compute confusion matrix
cm = confusion_matrix(y_actual, y_pred)

# Visualize the confusion matrix
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Spam', 'Spam'], yticklabels=['Not Spam', 'Spam'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Interpretation

- **A high number of True Positives (TP) and True Negatives (TN)** indicates a well-performing model.
- **A high number of False Positives (FP)** means too many **non-spam emails are incorrectly classified as spam**.
- **A high number of False Negatives (FN)** means **actual spam emails are not detected**, which could be risky in fraud detection or medical diagnosis.

ROC Curve (Receiver Operating Characteristic)

The **ROC Curve** is a graphical representation of a classification model's performance. It plots:

True Positive Rate (TPR) (Sensitivity or Recall)

Measures how many actual positive instances were correctly classified.

False Positive Rate (FPR)

Measures how many actual negative instances were incorrectly classified as positive.

The closer the **ROC Curve** is to the **top-left corner**, the better the model.

AUC Score (Area Under the Curve)

- **AUC = 1.0** → Perfect Model
- **AUC = 0.5** → Random Guessing (No discrimination ability)
- **AUC < 0.5** → Worse than random guessing

For example, an **AUC score of 0.90** means the model is **90% effective at distinguishing between positive and negative classes**.

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Example actual labels and predicted probabilities
y_test = [0, 0, 1, 1]  # Actual values
y_scores = [0.1, 0.4, 0.35, 0.8]  # Predicted probabilities

# Compute ROC curve
fpr, tpr, _ = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')  # Random guessing line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

Interpretation

- The **blue curve** represents the **actual model performance**.

- The **diagonal gray line** represents **random guessing**.
- **If the blue curve is above the gray line**, the model performs **better than random**.
- **A higher AUC score (closer to 1.0)** indicates a **stronger classification ability**.

Assignment Tasks

1. Support Vector Machines (SVM) – Decision Boundaries

Given a dataset where classes are not linearly separable, explain how SVM can still find an optimal decision boundary. Discuss the role of the Kernel Trick and compare the performance of Linear, Polynomial, and RBF kernels on high-dimensional data.
Bonus: Implement an SVM model using the dataset provided and compare accuracy with different kernels.

2. Random Forest – Overfitting vs. Generalization

Explain how Random Forest mitigates overfitting compared to a single Decision Tree. Discuss the significance of Bootstrapping and Feature Selection in improving model performance.
Practical Task: Train a Random Forest model on the provided dataset and analyze feature importance. Which features contribute most to loan approval?

3. Confusion Matrix and ROC Curve – Model Evaluation

A credit scoring model has the following confusion matrix for predicting high default risk:

Predicted ↓ / Actual →	Default (1)	No Default (0)
Default (1)	50	30
No Default (0)	10	110

- Calculate **Accuracy, Precision, Recall, and F1-Score** for this model.
- Draw the **ROC Curve** for this model assuming the model provides probability scores.
- **Interpretation:** What trade-offs are observed in Precision vs. Recall? How would you adjust the threshold to reduce false positives?

4. K-Nearest Neighbors (KNN) – Choosing Optimal K

In KNN, selecting the appropriate number of neighbors (K) is crucial for achieving a good balance between bias and variance. Explain the consequences of choosing a very small vs. very large K.
Task: Implement KNN on the given dataset and use **cross-validation** to determine the

optimal K. Analyze how accuracy varies with different values of K.

5. ### Bayesian Belief Networks (BBN) – Probabilistic Decision Making

   A bank uses a Bayesian Belief Network to assess the probability of loan default based on factors such as income, credit score, and previous defaults.

   - Explain how **Conditional Independence** can simplify the probability calculations in this network.
   - Given **prior probabilities**, construct a simple **Bayesian Network** with 3 nodes: **Loan Approval, Income Level, and Credit Score**.
   - **Implementation Challenge:** Use pgmpy in Python to model a **Bayesian Network** for loan approval. Compute the probability of approval given a high income but a low credit score.

Bonus Challenge

Compare SVM, Random Forest, and KNN on the dataset in terms of accuracy, precision, and recall. Which model is the most robust for predicting loan approval and why?
These **challenging questions** will test both **conceptual understanding and practical implementation** of **machine learning models, evaluation techniques, and Bayesian reasoning**.