Machine Learning

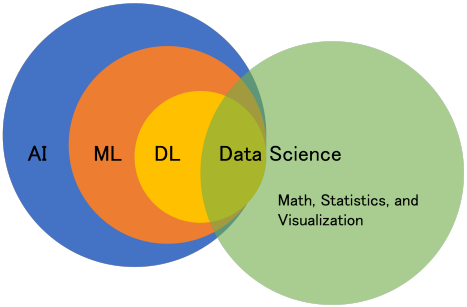
Thien Huynh-The

Department of Computer and Communications Engineering
HCMC University of Technology and Education

December 26, 2025

AI, ML, and DL: Definitions

- **Artificial Intelligence (AI):** Machines performing tasks requiring human intelligence. Encompasses rule-based systems and advanced machine learning.
- **Machine Learning (ML):** A subfield of AI. Systems learn from data without explicit programming. Algorithms learn patterns and make predictions. Examples: Linear Regression, SVMs.
- **Deep Learning (DL):** A subfield of ML using deep neural networks (multiple layers) to analyze data and learn complex patterns. Examples: CNNs, RNNs.



Machine Learning: Learning from Experience

- **Definition:** 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.
- **Example: Email Spam Filtering**
  - **T:** Classifying emails as "spam" or "not spam."
  - **P:** Accuracy in classifying emails.
  - **E:** A dataset of emails labeled as "spam" or "not spam." The ML system learns patterns from this labeled data to improve its spam detection accuracy.

Learning from Experience: Task

T (Tasks): The specific problems the ML system is designed to solve. Examples:

- *Classification:* Assigning an input to one of several predefined categories (e.g., spam filtering, image classification).
- *Regression:* Predicting a continuous value (e.g., house prices, stock prices).
- *Clustering:* Grouping similar inputs together (e.g., customer segmentation).

Tasks: Classification, Regression, and Clustering

Feature	Classification	Regression	Clustering
Task	Assign an input to one of a set of pre-defined categories.	Predict a continuous numerical value.	Group similar inputs together into clusters.
Output	Discrete class label (e.g., "spam," "not spam," "cat," "dog").	Continuous numerical value (e.g., house price, temperature).	Assignment of each input to a specific cluster.
Training Data	Labeled data (input-output pairs).	Labeled data (input-output pairs).	Unlabeled data (inputs only).
Goal	Maximize the accuracy of classification.	Minimize the error between predicted and actual values.	Maximize similarity within clusters and minimize similarity between clusters.
Examples	Email spam detection, image classification (cat vs. dog), medical diagnosis.	House price prediction, stock price forecasting, temperature prediction.	Customer segmentation, document categorization, anomaly detection.
Algorithms	Logistic Regression, Support Vector Machines (SVMs), Decision Trees, Random Forests, Naive Bayes.	Linear Regression, Polynomial Regression, Support Vector Regression (SVR), Neural Networks.	K-Means, DBSCAN, Hierarchical Clustering, Gaussian Mixture Models (GMMs).

Table: Comparison of ML Tasks

Tasks: Classification, Regression, and Clustering

Key Difference: Data Labels and Learning Paradigm

- **Supervised Learning:**
  - Uses *labeled data* (input-output pairs).
  - Learns a mapping from inputs to outputs to predict new outputs.
  - Examples:
    - Classification (e.g., spam detection).
    - Regression (e.g., house price prediction).
- **Unsupervised Learning:**
  - Uses *unlabeled data* (inputs only).
  - Finds patterns or structure in the data (e.g., clustering).
  - Example: Customer segmentation.

Learning from Experience: Experience

The "experience" in machine learning refers to the data used to train the model. The nature and quality of this data are crucial for the model's performance.

Data Types:

- Labeled Data: Each data point is associated with a corresponding output or label. Used in supervised learning.
  - Examples: (input, output) pairs like (image, "cat"), (email, "spam"), (house features, price).
  - Use Cases: Classification, Regression.
- Unlabeled Data: Data points without any associated outputs or labels. Used in unsupervised learning.
  - Examples: A collection of images without labels, customer purchase histories.
  - Use Cases: Clustering, dimensionality reduction, anomaly detection.

Learning from Experience: Experience (cont

Data Characteristics: The quality of the experience (data) significantly impacts the learning process. Important characteristics include:

- Quantity: More data generally leads to better performance, especially for complex models.
- Quality: Clean, accurate, and consistent data is essential. Noisy or erroneous data can hinder learning.
- Relevance: The data should be relevant to the task being learned. Irrelevant data can confuse the model.
- Bias: Data can contain biases that reflect existing societal or other prejudices. These biases can be learned by the model, leading to unfair or inaccurate predictions.
- Representation: The data should be representative of the real-world scenarios the model will encounter. If the training data is not representative, the model may not generalize well to new data.

Learning from Experience: Performance

P (Performance Measure): A metric used to evaluate how well the system performs on the tasks. Examples:

- Accuracy: The proportion of correctly classified instances.
- Precision: The proportion of true positives among the predicted positives.
- Recall: The proportion of true positives among the actual positives.
- Mean Squared Error (MSE): Average squared difference between predicted and actual values (for regression).

Performance Metrics for Classification

Confusion Matrix: A table summarizing the performance of a classification model.

	Predicted	
Actual	Positive	Negative
Positive	True Positive (TP)	False Negative (FN)
Negative	False Positive (FP)	True Negative (TN)

- **Accuracy:** Fraction of correctly classified instances.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision:** Fraction of true positives among predicted positives.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Performance Metrics for Classification (cont)

Confusion Matrix: A table summarizing the performance of a classification model.

	Predicted	
Actual	Positive	Negative
Positive	True Positive (TP)	False Negative (FN)
Negative	False Positive (FP)	True Negative (TN)

- **Recall (Sensitivity):** Fraction of true positives among actual positives.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score:** Harmonic mean of precision and recall.

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Performance Metrics for Regression

- **Mean Squared Error (MSE):** Average squared difference between predicted and actual values.

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

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of data points.

- **Root Mean Squared Error (RMSE):** Square root of MSE.

$$\text{RMSE} = \sqrt{\text{MSE}}$$

- **Mean Absolute Error (MAE):** Average absolute difference between predicted and actual values.

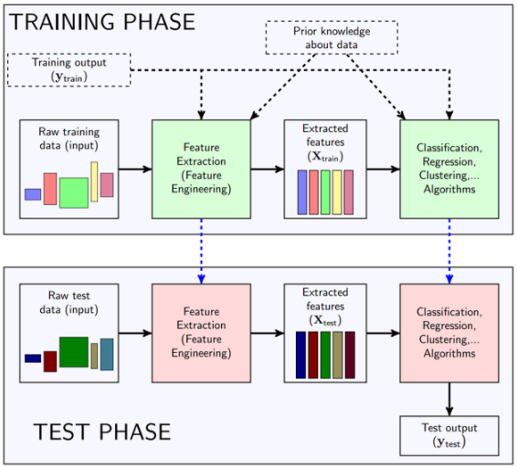
$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **R-squared (Coefficient of Determination):** Represents the proportion of variance in the dependent variable that is predictable from the independent variables.

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

where $\bar{y}$ is the mean of the actual values.

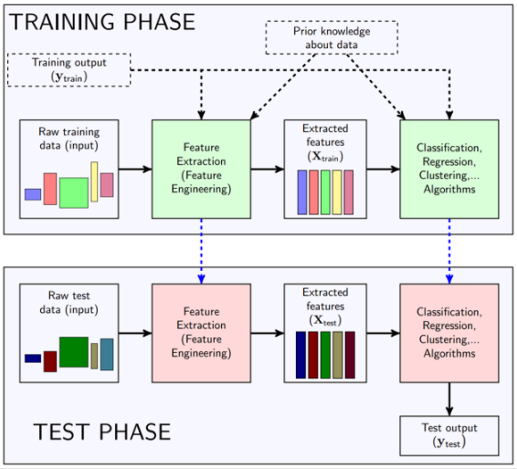
Overview of a ML Framework



Training Phase:

- Data Collection: Gathering raw training data.
- Feature Extraction/Engineering: Transforming raw data into meaningful features (X_{train}). Prior knowledge can be incorporated.
- Model Training: Applying a learning algorithm (Classification, Regression, Clustering, etc.) to X_{train} and y_{train} to train a model.

Overview of a ML Framework



Test Phase:

- Data Collection: Gathering raw test data.
- Feature Extraction/Engineering: Applying the same feature extraction to raw test data to get X_{test}.
- Model Evaluation/Prediction: Using the trained model to predict on X_{test} and get y_{test}.

Key Considerations:

- Feature engineering is crucial.
- Test data should be representative.
- Model evaluation is essential.

Feature Engineering: Introduction

What is Feature Engineering?

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data. It is a crucial step in machine learning, often more impactful than the choice of model itself.

- **Why is it important?**
  - Improves model performance.
  - Simplifies model complexity.
  - Enhances model interpretability.
- **Two Main Components:**
  - Feature Extraction
  - Feature Selection

Feature Extraction

Feature Extraction: Creating New Features

Feature extraction involves creating new features from raw data. This can involve:

- **Manual Feature Engineering:** Using domain expertise to create features.
- **Automated Feature Extraction:** Using algorithms to automatically generate features.
- **Examples:**
  - Text Data: Bag-of-words, TF-IDF, word embeddings.
  - Image Data: Edge detection, SIFT, HOG, CNN features.
  - Time Series Data: Rolling averages, Fourier transforms, autocorrelation.
- **Dimensionality Reduction:** Feature extraction can also be used to reduce the dimensionality of the data while preserving important information (e.g., PCA).

Feature Extraction: Time Domain Examples

Time domain features are calculated directly from the raw time series data. They capture characteristics of the signal's amplitude and variation over time.

- **Mean:** Average value of the signal.

$$\text{Mean} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

- **Standard Deviation (Std Dev):** Measure of the signal's dispersion around the mean.

$$\text{Std Dev} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \text{Mean})^2}$$

- **Root Mean Square (RMS):** Measure of the signal's magnitude.

$$\text{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}$$

- **Zero-Crossing Rate:** Number of times the signal crosses zero.
- **Peak-to-Peak Amplitude:** Difference between the maximum and minimum values of the signal.

Feature Extraction: Frequency Domain Examples

Frequency domain features are calculated after transforming the signal into the frequency domain using techniques like the Fourier Transform, reveal its frequency content.

- **Discrete Fourier Transform (DFT):** Decomposes the signal into its constituent frequencies.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi kn/N}$$

where x_n is the time domain signal, X_k is the frequency domain representation, and j is the imaginary unit.

- **Power Spectral Density (PSD):** Describes how the power of a signal is distributed across different frequencies.

$$\text{PSD} = \frac{|X_k|^2}{N}$$

- **Dominant Frequency:** The frequency with the highest power in the PSD.
- **Spectral Centroid:** The "center of mass" of the signal's power spectrum (where f_k are the frequencies).

$$\text{Spectral Centroid} = \frac{\sum_{k=1}^{N/2} f_k |X_k|^2}{\sum_{k=1}^{N/2} |X_k|^2}$$

- **Spectral Bandwidth:** Measure of the width of the frequency band occupied by the signal.

Feature Selection

Feature Selection: Choosing the Best Features

Feature selection involves selecting a subset of the most relevant features from the existing set of features. This can help to:

- Improve model performance by reducing overfitting.
- Simplify the model, making it faster and easier to interpret.
- Reduce computational cost.
- **Methods:**
  - Filter Methods: Select features based on statistical measures (e.g., correlation, chi-squared test).
  - Wrapper Methods: Evaluate subsets of features by training a model on them (e.g., recursive feature elimination).
  - Embedded Methods: Feature selection is done as part of the model training process (e.g., LASSO regularization).

Feature Selection: Filter Methods

Filter methods select features based on statistical measures calculated from the data. They evaluate each feature independently of the chosen learning algorithm.

- **Key Characteristics:**
  - Evaluate features independently.
  - Computationally efficient.
  - Do not consider the interaction with the learning algorithm.
  - Prone to selecting redundant features.
- **Common Metrics/Methods:**
  - **Correlation:** Measures the linear relationship between two variables.

$$\text{Correlation}(X, Y) = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y}$$

where $\text{Cov}(X, Y)$ is the covariance between X and Y, and σ_X and σ_Y are the standard deviations of X and Y, respectively. Select features with high correlation to the target variable.

Feature Selection: Filter Methods

- **Common Metrics/Methods:**
  - **Chi-Squared Test (χ^2):** Measures the independence between categorical variables. Used for feature selection in classification problems with categorical features.

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

    where O_i are the observed frequencies and E_i are the expected frequencies. Select features with high χ^2 values (indicating dependence on the target variable).
  - **Analysis of Variance:** In regression with categorical features, ANOVA selects features with significantly higher variance *between* groups than *within* groups.
  - **Information Gain/Mutual Information:** Selects features that provide the greatest reduction in uncertainty (entropy) about the target variable.
- Advantages: Computationally fast, scalable to high-dimensional datasets.
- Disadvantages: Ignores feature dependencies, may select redundant features.

Other Feature Engineering Techniques

Beyond Extraction and Selection

Besides extraction and selection, other important feature engineering techniques include:

- **Data Cleaning:** Handling missing values, outliers, and inconsistencies.
- **Data Transformation:** Scaling, normalization, and encoding categorical variables.
  - Scaling: Standardizing or normalizing numerical features.
  - Encoding: Converting categorical variables into numerical representations (e.g., one-hot encoding, label encoding).
- **Feature Creation/Construction:** Combining existing features to create new ones (e.g., creating interaction terms).

Data Transformation: Scaling

Scaling: Bringing Features to a Similar Scale

Scaling transforms numerical features to a similar range of values. This is important because features with larger values can dominate those with smaller values, which can negatively impact the performance of some machine learning algorithms.

- **Why is scaling important?**
  - Prevents features with larger magnitudes from dominating others.
  - Improves the convergence speed of gradient-based optimization algorithms.
  - Improves the performance of distance-based algorithms (e.g., k-NN, SVM).

Data Transformation: Scaling

Common Scaling Techniques:

- **Min-Max Scaling (Normalization):** Scales features to a range between 0 and 1.

$$x' = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

where x is the original value, x_{min} is the minimum value, and x_{max} is the maximum value of the feature.

- **Standardization (Z-score normalization):** Scales features to have a mean of 0 and a standard deviation of 1.

$$x' = \frac{x - \mu}{\sigma}$$

where x is the original value, μ is the mean, and σ is the standard deviation of the feature.

- **Robust Scaling:** Scales features using median and interquartile range (IQR), making it less sensitive to outliers.

$$x' = \frac{x - \text{Median}}{\text{IQR}}$$

where IQR is the interquartile range (Q3-Q1).

Feature Engineering: Summary

Key Takeaways

- Feature engineering is a crucial step in building effective machine learning models.
- It involves transforming raw data into meaningful features through extraction, selection, cleaning, transformation, and creation.
- Domain expertise is often essential for effective feature engineering.
- It's an iterative process that requires experimentation and evaluation.

In-Class Assignment

Topics Covered:

- Discovery evaluation protocols:
  - k-fold cross validation
  - Leave one out cross validation
- Feature selection methods:
  - Wrapper method
  - Embedded method
- Other Engineering techniques:
  - Encoding method

Introduction to Regression

Regression: Predicting Continuous Values

Regression is a supervised learning task where the goal is to predict a continuous numerical output (target variable) based on input features.

- **Examples:**
  - Predicting house prices based on features like size, location, and number of bedrooms.
  - Forecasting stock prices based on historical data and market indicators.
  - Estimating temperature based on weather patterns.
- **Types of Regression:**
  - Linear Regression
  - Polynomial Regression
  - Support Vector Regression (SVR)
  - Decision Tree Regression
  - Random Forest Regression
  - Neural Network Regression

Linear Regression

Linear Regression: Modeling Linear Relationships

Linear regression assumes a linear relationship between the input features and the target variable.

- **Simple Linear Regression:** One input feature.

$$y = \beta_0 + \beta_1 x$$

  where y is the predicted output, x is the input feature, β_0 is the intercept, and β_1 is the slope.

- **Multiple Linear Regression:** Multiple input features.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

  where $x_1, x_2, \ldots, x_n$ are the input features, and $\beta_1, \beta_2, \ldots, \beta_n$ are their corresponding coefficients.

- **Goal:** Find the optimal values for the coefficients (βs) that minimize the difference between the predicted and actual values (e.g., using Mean Squared Error).

Other Regression Methods

Beyond Linear Relationships

When the relationship between the input features and the target variable is non-linear, other regression methods are more appropriate.

- **Polynomial Regression:** Models non-linear relationships using polynomial functions.
- **Support Vector Regression (SVR):** Uses support vectors to define a margin of tolerance around the predicted values.
- **Decision Tree Regression:** Partitions the feature space into regions and predicts a constant value within each region.
- **Random Forest Regression:** An ensemble method that combines multiple decision trees to improve prediction accuracy.
- **Neural Network Regression:** Uses neural networks to learn complex non-linear relationships.

Linear Regression: Introduction

Linear regression models the relationship between a dependent variable y and one or more independent variables x using a linear equation.

- **Types of Linear Regression:**
  - **Simple Linear Regression:** One independent variable.

$$y = \beta_0 + \beta_1 x$$

  - **Multiple Linear Regression:** Multiple independent variables.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

- **Goal:** Find the optimal coefficients (βs) that minimize the error between predicted and actual values.

Data Preprocessing for Linear Regression

Several preprocessing steps are often necessary to ensure the best performance of a linear regression model:

- **Handling Missing Values:** Imputation (mean, median), removal.
- **Outlier Detection and Treatment:** Removal, transformation (e.g., log transformation).
- **Feature Scaling:** Important when features have different scales (e.g., standardization, min-max scaling).
- **Feature Encoding:** Converting categorical variables into numerical representations (e.g., one-hot encoding).
- **Feature Selection/Engineering:** Selecting relevant features and creating new ones.

Evaluating Linear Regression: Mean Squared Error (MSE)

Mean Squared Error (MSE): A Common Evaluation Metric

MSE measures the average squared difference between the predicted and actual values.

- **Formula:**

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

  where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of data points.

- **Interpretation:**
  - A lower MSE indicates better model performance.
  - MSE is sensitive to outliers due to the squared term.
- **Other Metrics:** RMSE (Root Mean Squared Error), MAE (Mean Absolute Error), R-squared.

Assumptions of Linear Regression

Key Assumptions

Linear regression models are based on several assumptions. Violating these assumptions can affect the model's validity.

- **Linearity:** The relationship between the independent and dependent variables is linear.
- **Independence of Errors:** The errors (residuals) are independent of each other.
- **Homoscedasticity:** The variance of the errors is constant across all levels of the independent variables.
- **Normality of Errors:** The errors are normally distributed.
- **No Multicollinearity:** The independent variables are not highly correlated with each other.

Estimating Weights in Linear Regression

Finding the Best Fit Line

In linear regression, we aim to find a function that best describes the linear relationship between input features (x) and the output (y):

$$\hat{y} = f(x) = w_1 x_1 + w_0 = \mathbf{x}^T \mathbf{w}$$

where:

- $\hat{y}$ is the predicted output.
- $\mathbf{x} = \begin{bmatrix} x_1 \\ 1 \end{bmatrix}$ is the input vector (including a constant 1 for the bias term).
- $\mathbf{w} = \begin{bmatrix} w_1 \\ w_0 \end{bmatrix}$ is the weight vector (w_1 is the weight for x_1, and w_0 is the bias).

Loss Function and Minimization

Minimizing the Error

We want to minimize the error between the predicted values ($\hat{y}$) and the actual values (y). A common way to do this is by minimizing the Mean Squared Error (MSE). The loss function is:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - \mathbf{x}_i^T \mathbf{w})^2$$

where:

- N is the number of data points.
- y_i is the actual output for the i-th data point.
- $\mathbf{x}_i$ is the input vector for the i-th data point.

We want to find the weights $\mathbf{w}$ that minimize this loss:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{L}(\mathbf{w})$$

Matrix Form and Solution

Matrix Form and Normal Equation

The loss function can be expressed in matrix form:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2N}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

where $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$ is the vector of actual outputs, $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}$ is the design matrix (each row is an input vector).

To find the minimum, we take the derivative of the loss function with respect to $\mathbf{w}$ and set it to zero:

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{N}\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

Solving for $\mathbf{w}$ gives the normal equation:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Handling Non-Invertibility

Pseudo-Inverse
If $\mathbf{X}^T\mathbf{X}$ is not invertible (singular), we use the pseudo-inverse:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^\dagger \mathbf{X}^T\mathbf{y}$$

where $(\mathbf{X}^T\mathbf{X})^\dagger$ is the Moore-Penrose pseudo-inverse of $\mathbf{X}^T\mathbf{X}$.

The pseudo-inverse always exists and provides the least-squares solution, minimizing the squared error even when the matrix is not invertible.

Linear Regression Example: Data Preparation

Data:

- Input (Height - $\mathbf{x}$)
- Output (Weight - $\mathbf{y}$)

$$\mathbf{x} = \begin{bmatrix} 147 \\ 150 \\ 153 \\ 155 \\ 158 \\ 160 \\ 163 \\ 165 \\ 168 \\ 170 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 49 \\ 50 \\ 51 \\ 52 \\ 54 \\ 56 \\ 58 \\ 59 \\ 60 \\ 62 \end{bmatrix}$$

Linear Regression Example: Calculating X^TX and $(X^TX)^{-1}$

Calculations:

- **Design Matrix (X):** Adding a column of ones for the bias term:

$$\mathbf{X} = \begin{bmatrix} 147 & 1 \\ 150 & 1 \\ 153 & 1 \\ 155 & 1 \\ 158 & 1 \\ 160 & 1 \\ 163 & 1 \\ 165 & 1 \\ 168 & 1 \\ 170 & 1 \end{bmatrix}$$

Linear Regression Example: Calculating X^TX and $(X^TX)^{-1}$

Calculations:

- $\mathbf{X}^T\mathbf{X}$:

$$\mathbf{X}^T\mathbf{X} = \begin{bmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & N \end{bmatrix} = \begin{bmatrix} 252765 & 1599 \\ 1599 & 10 \end{bmatrix}$$

- $(\mathbf{X}^T\mathbf{X})^{-1}$:

$$(\mathbf{X}^T\mathbf{X})^{-1} \approx \begin{bmatrix} 0.1656 & -26.477 \\ -26.477 & 4188.72 \end{bmatrix}$$

Linear Regression Example: Calculating $X^T y$ and w

Calculations:

- $\mathbf{X}^T\mathbf{y}$:

$$\mathbf{X}^T\mathbf{y} = \begin{bmatrix} \sum x_i y_i \\ \sum y_i \end{bmatrix} = \begin{bmatrix} 81077 \\ 541 \end{bmatrix}$$

- **Weights (w):**

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \approx \begin{bmatrix} 0.612 \\ -41.31 \end{bmatrix}$$

Linear Regression Example: The Equation and Prediction

The Linear Regression Equation:

$$\hat{y} = 0.612x - 41.31$$

Example Prediction (Height = 160):

$$\hat{y} = 0.612 \times 160 - 41.31 \approx 56.61$$

Conclusion:
A person with a height of 160 cm is predicted to have a weight of approximately 56.61 kg.

Evaluating Linear Regression: Predictions

New Test Data:

- Height ($\mathbf{x}_{test}$)
- Actual Weight ($\mathbf{y}_{test}$)

$$\mathbf{x}_{test} = \begin{bmatrix} 173 \\ 175 \\ 178 \\ 180 \\ 183 \end{bmatrix}, \mathbf{y}_{test} = \begin{bmatrix} 63 \\ 64 \\ 66 \\ 67 \\ 68 \end{bmatrix}$$

Predictions ($\hat{\mathbf{y}}_{test}$): Applying the equation to each height:

$$\hat{y}_1 = 0.612 \times 173 - 41.31 \approx 64.476$$
$$\hat{y}_2 = 0.612 \times 175 - 41.31 \approx 65.696$$
$$\hat{y}_3 = 0.612 \times 178 - 41.31 \approx 67.536$$
$$\hat{y}_4 = 0.612 \times 180 - 41.31 \approx 68.756$$
$$\hat{y}_5 = 0.612 \times 183 - 41.31 \approx 70.596$$

Evaluating Linear Regression: Mean Squared Error (MSE)

Calculating MSE:

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

$$\begin{aligned}
\text{MSE} &= \frac{1}{5}[(63 - 64.476)^2 + (64 - 65.696)^2 + (66 - 67.536)^2 \\
&\quad + (67 - 68.756)^2 + (68 - 70.596)^2] \\
&\approx \frac{1}{5}[2.179 + 2.876 + 2.350 + 3.084 + 6.749] \\
&\approx \frac{1}{5}[17.238] \\
&\approx 3.448
\end{aligned}$$

Result: The Mean Squared Error for this test data is approximately 3.448.

Linear Regression: Calculation Summary

Given input data $\mathbf{x}$ and corresponding outputs $\mathbf{y}$, the weights $\mathbf{w}$ for the linear regression equation $\hat{y} = \mathbf{x}^T\mathbf{w}$ can be calculated as follows:

1. **Prepare the Data:**
   - Create the design matrix $\mathbf{X}$ by adding a column of ones to $\mathbf{x}$ for the bias term.
   - Ensure $\mathbf{x}$ and $\mathbf{y}$ are column vectors.
2. **Calculate $\mathbf{X}^T\mathbf{X}$:** Multiply the transpose of $\mathbf{X}$ by $\mathbf{X}$.
3. **Calculate the Inverse of $\mathbf{X}^T\mathbf{X}$:** Calculate $(\mathbf{X}^T\mathbf{X})^{-1}$. If $\mathbf{X}^T\mathbf{X}$ is singular (not invertible), use the pseudo-inverse $(\mathbf{X}^T\mathbf{X})^{\dagger}$.
4. **Calculate $\mathbf{X}^T\mathbf{y}$:** Multiply the transpose of $\mathbf{X}$ by $\mathbf{y}$.
5. **Calculate the Weights $\mathbf{w}$:**
$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$
(or $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{\dagger}\mathbf{X}^T\mathbf{y}$ if using the pseudo-inverse).
6. **The Linear Regression Equation:** The equation is then $\hat{y} = \mathbf{x}^T\mathbf{w}$, or in scalar form (for simple linear regression): $\hat{y} = w_1 x + w_0$.

Key Points:

- The pseudo-inverse is used when $(\mathbf{X}^T\mathbf{X})$ is not invertible.
- This method minimizes the sum of squared errors.

Python Code - Manual Calculation

```
from __future__ import division, print_function, unicode_literals
import numpy as np
import matplotlib.pyplot as plt

# height (cm)
X = np.array([[147, 150, 153, 155, 158, 160, 163, 165, 168, 170]]).T
# weight (kg)
y = np.array([[49, 50, 51, 52, 54, 56, 58, 59, 60, 62]]).T
# Visualize data
plt.plot(X, y, 'ro')
plt.axis([140, 190, 45, 75])
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.show()
```

Python Code - Manual Calculation

```
# Building Xbar
one = np.ones((X.shape[0], 1))
Xbar = np.concatenate((one, X), axis = 1)
# Calculating weights of the fitting line
A = np.dot(Xbar.T, Xbar)
b = np.dot(Xbar.T, y)
w = np.dot(np.linalg.pinv(A), b)
print('w = ', w)
# Preparing the fitting line
w_0 = w[0][0]
w_1 = w[1][0]
x0 = np.linspace(145, 185, 2)
y0 = w_0 + w_1*x0
```

Python Code - Manual Calculation

```
# Drawing the fitting line
plt.plot(X.T, y.T, 'ro')     # data
plt.plot(x0, y0)             # the fitting line
plt.axis([140, 190, 45, 75])
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.show()
```

Python Code - Using Library

```
from sklearn import datasets, linear_model
# fit the model by Linear Regression
regr = linear_model.LinearRegression(fit_intercept=False)
# fit_intercept = False for calculating the bias
regr.fit(Xbar, y)

# Compare two results
print( 'Solution found by scikit-learn  : ', regr.coef_ )
print( 'Solution found by (5): ', w.T)
```

Student Assignment: Code Comparison

Objective: To compare the results of manual linear regression calculations with those obtained using the scikit-learn library.

- Students will implement linear regression from scratch using matrix operations (as derived in class).
- Students will then use scikit-learn's `LinearRegression` class to fit the same data.
- By comparing the resulting weight vectors, students will:
  - Verify the correctness of their manual calculations.
  - Gain practical experience with a widely used machine learning library.
  - Develop a deeper understanding of the underlying mathematical principles of linear regression.

Introduction to Classification

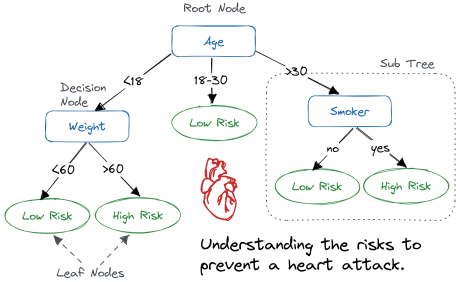
Classification is a supervised learning task where the goal is to predict a categorical output (a class label) based on input features.

- **Examples:**
  - Email spam detection (spam/not spam).
  - Image classification (cat/dog/bird).
  - Medical diagnosis (disease/no disease).
- **Common Classification Algorithms:**
  - Decision Trees
  - Random Forests
  - Support Vector Machines (SVMs)
  - k-Nearest Neighbors (k-NN)
  - Neural Networks
  - Logistic Regression (often used for binary classification)

Decision Trees

Decision trees create a tree-like structure of decisions based on feature values. Each internal node represents a feature, each branch represents a decision rule, and each leaf node represents an outcome (class label).

- **How it works:** Recursively partitions the data based on feature values to create homogeneous subsets (i.e., subsets with mostly the same class label).
- **Advantages:** Easy to understand and interpret, can handle both categorical and numerical data.
- **Disadvantages:** Prone to overfitting, can be sensitive to small changes in the data.



Random Forests

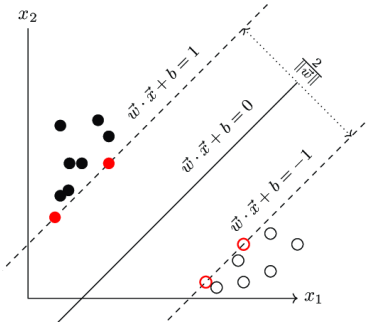
Random forests are an ensemble learning method that combines multiple decision trees to improve prediction accuracy and reduce overfitting.

- **How it works:** Creates multiple decision trees on random subsets of the data and random subsets of features. The final prediction is made by aggregating the predictions of all trees (e.g., by majority voting).
- **Advantages:** High accuracy, robust to overfitting, handles high-dimensional data well.
- **Disadvantages:** More complex than single decision trees, less interpretable.

Support Vector Machines (SVMs)

SVMs find the optimal hyperplane that best separates data points of different classes.

- **How it works:** Maps data points to a high-dimensional space and finds a hyperplane with the largest margin between classes. Can use kernel functions to handle non-linearly separable data.
- **Advantages:** Effective in high-dimensional spaces, relatively memory efficient.
- **Disadvantages:** Can be computationally intensive for large datasets, sensitive to kernel choice and hyperparameters.



k-Nearest Neighbors (k-NN)

k-NN: Classifying Based on Neighbors

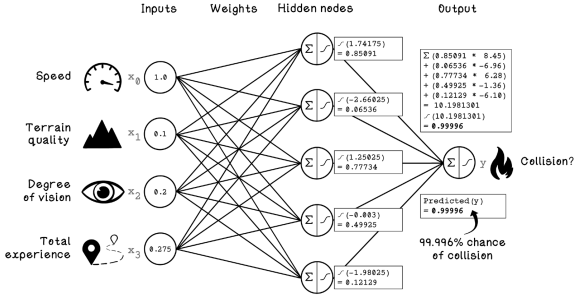
k-NN classifies a data point based on the majority class among its k nearest neighbors in the feature space.

- **How it works:** Calculates the distance between the new data point and all training data points. Selects the k nearest neighbors and assigns the most frequent class among them.
- **Advantages:** Simple to implement, no training phase.
- **Disadvantages:** Computationally expensive for large datasets, sensitive to feature scaling and the choice of k.

Neural Networks

Neural networks are complex models inspired by the structure of the human brain. They consist of interconnected layers of neurons that learn complex non-linear relationships in the data.

- **How it works:** Multiple layers of interconnected nodes (neurons) process information through weighted connections. Learning occurs by adjusting these weights based on the data.
- **Advantages:** Can learn highly complex patterns, achieve high accuracy on various tasks.
- **Disadvantages:** Can be computationally expensive to train, require large amounts of data, can be difficult to interpret (black box).



Decision Trees: Introduction and Nominal Data

Decision Trees: A Tree-Based Approach to Classification

Decision trees are a powerful and interpretable classification method that uses a tree-like structure to represent decisions and their possible outcomes.

Nominal Data:

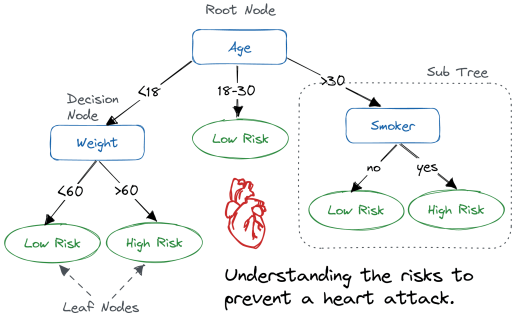
- Nominal data represents categories or labels with no inherent order.
- Examples: Colors (red, blue, green), types of fruit (apple, banana, orange), weather conditions (sunny, cloudy, rainy).
- Used as attributes (features) in decision trees to split the data.

Decision Trees: Concept and Principle

A decision tree consists of:

- **Nodes:** Represent decisions based on feature values.
- **Branches:** Represent the outcomes of the decisions.
- **Leaves:** Represent the final outcomes or class labels.

The principle behind decision tree construction is to recursively partition the data into subsets that are as "pure" as possible with respect to the target variable. "Purity" means that the subsets contain mostly instances of a single class.



Building a Decision Tree: Picking the Root Attribute

The core of decision tree construction lies in selecting the most informative attribute for splitting the data at each node. This process aims to create subsets of data that are increasingly "pure" (i.e., contain instances primarily of a single class).

- **The Goal:** To maximize the separation between classes after the split.
- **How it Works:**
  - For each attribute:
    - Calculate the entropy of the current node (before the split).
    - Calculate the entropy of each child node that would result from splitting on that attribute.
    - Calculate the information gain achieved by splitting on that attribute.
  - Select the attribute with the **highest information gain** as the splitting attribute for the current node.
- **Example:** With fruit data (apple, banana, orange) and features like color (red, yellow, orange) and size (small, medium, large), we calculate information gain for each feature. The feature with higher gain is chosen for the split.
- **Recursive Process:** This splitting process repeats for each child node until a stopping criterion is met (e.g., pure nodes or maximum depth).

Entropy

Entropy measures the impurity or disorder of a set of data. In the context of classification, it measures the degree to which a set of instances is mixed with different class labels.

- **Formula:**

$$H(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

where:

- S is the set of instances.
- c is the number of classes.
- p_i is the proportion of instances in S that belong to class i.

- **Interpretation:**
  - $H(S) = 0$ if all instances belong to the same class (pure set).
  - $H(S)$ is maximum when the instances are evenly distributed across all classes (most impure set).

Information Gain

Information gain measures the reduction in entropy achieved by splitting the data on a particular attribute.

- **Formula:**

$$\text{Gain}(S, A) = H(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} H(S_v)$$

  where:
  - S is the set of instances.
  - A is the attribute.
  - $\text{Values}(A)$ is the set of possible values of attribute A.
  - S_v is the subset of S where attribute A has value v.
- **Selecting the Root Node:** The attribute with the highest information gain is chosen as the root node for splitting.

Decision Tree Example: Data

Outlook	Temp.	Humidity	Wind	Play?
S	H	H	W	-
S	H	H	S	-
O	H	H	W	+
R	M	H	W	+
R	C	N	W	+
R	C	N	S	-
O	C	N	S	+
S	M	H	W	-
S	C	N	W	+
R	M	N	W	+
S	M	N	S	+
O	M	H	S	+
O	H	N	W	+
R	M	H	S	-

- Outlook
  - S(unny)
  - O(vercast)
  - R(ainy)
- Temperature
  - H(ot)
  - M(edium)
  - C(ool)
- Humidity
  - H(igh)
  - N(ormal)
  - L(ow)
- Wind
  - S(trong)
  - W(eak)

Decision Tree Example: Initial Entropy

Initial Entropy (S):

- Total instances (N) = 14
- Positive instances (+) = 9
- Negative instances (-) = 5
- Calculation

$$p(+) = \frac{9}{14} \simeq 0.643$$
$$p(-) = \frac{5}{14} \simeq 0.357$$
$$H(S) = -p(+)\log_2(p(+)) - p(-)\log_2(p(-)) \simeq 0.940$$

Decision Tree Example: Information Gain for Outlook

Information Gain for Outlook:

- Outlook = Sunny (S): H(S⟨sub⟩Sunny⟨/sub⟩) $\simeq 0.971$ (5 instances: 2+, 3-)
- Outlook = Overcast (O): H(S⟨sub⟩Overcast⟨/sub⟩) $= 0$ (4 instances: 4+, 0-)
- Outlook = Rainy (R): H(S⟨sub⟩Rainy⟨/sub⟩) $\simeq 0.971$ (5 instances: 3+, 2-)

where:

- H(): Represents the entropy function.
- S⟨sub⟩Sunny⟨/sub⟩: Represents the subset of the data where the Outlook is "Sunny."
- S⟨sub⟩Overcast⟨/sub⟩: Represents the subset of the data where the Outlook is "Overcast."
- S⟨sub⟩Rainy⟨/sub⟩: Represents the subset of the data where the Outlook is "Rainy."

$$\begin{aligned}
\text{Gain}(S, \text{Outlook}) &= H(S) - \left(\frac{5}{14} H(S_{\text{Sunny}}) + \frac{4}{14} H(S_{\text{Overcast}}) + \frac{5}{14} H(S_{\text{Rainy}}) \right) \\
&\simeq 0.940 - \left(\frac{5}{14} \times 0.971 + \frac{4}{14} \times 0 + \frac{5}{14} \times 0.971 \right) \\
&\simeq 0.246
\end{aligned}$$

Decision Tree Example: Information Gain for Other Attributes

Information Gain for Other Attributes:

- **Temperature:**
  - $H(S_{\text{Hot}}) = 1$ (4 instances: 2+, 2-)
  - $H(S_{\text{Medium}}) \simeq 0.918$ (6 instances: 4+, 2-)
  - $H(S_{\text{Cool}}) \simeq 0.811$ (4 instances: 3+, 1-)

  $\text{Gain}(S, \text{Temperature}) \simeq 0.029$
- **Humidity:**
  - $H(S_{\text{High}}) \simeq 0.985$ (7 instances: 3+, 4-)
  - $H(S_{\text{Normal}}) \simeq 0.592$ (7 instances: 6+, 1-)

  $\text{Gain}(S, \text{Humidity}) \simeq 0.151$
- **Wind:**
  - $H(S_{\text{Strong}}) = 1$ (6 instances: 3+, 3-)
  - $H(S_{\text{Weak}}) \simeq 0.811$ (8 instances: 6+, 2-)

  $\text{Gain}(S, \text{Wind}) \simeq 0.048$

Decision Tree Example: Choosing the Root Node

Comparing Information Gains:

- Gain(S, Outlook) $\simeq 0.246$
- Gain(S, Temperature) $\simeq 0.029$
- Gain(S, Humidity) $\simeq 0.151$
- Gain(S, Wind) $\simeq 0.048$

Conclusion:

Outlook has the highest information gain. Therefore, **Outlook** is selected as the root node of the decision tree.

Building the Decision Tree: Next Steps

After determining "Outlook" as the root node, we proceed by creating subsets of the data based on its values (Sunny, Overcast, Rainy) and repeating the information gain calculation for each subset.

1. Create Subsets:

- Sunny Subset: Instances where Outlook = Sunny.
- Overcast Subset: Instances where Outlook = Overcast.
- Rainy Subset: Instances where Outlook = Rainy.

Building the Decision Tree: Splitting Each Subset

2. Splitting Each Subset:

For each subset created in the previous step:

1. Calculate the entropy of the subset $H(S_{\text{subset}})$.
2. For each remaining attribute (Temperature, Humidity, Wind):
   2.1 Calculate the information gain if we split the subset on that attribute: $\text{Gain}(S_{\text{subset}}, \text{Attribute})$.
3. Choose the attribute with the highest information gain as the splitting attribute for that branch.

Building the Decision Tree: Recursive Process and Example

3. Recursive Process:

The splitting process is repeated recursively for each newly created branch until a stopping criterion is met:

- **Stopping Criteria:**
  - All instances in a subset belong to the same class (pure node).
  - No more attributes to split on.
  - A predefined maximum tree depth is reached.
  - The number of instances in a node falls below a threshold.

Outlook Subset

Example: Splitting the "Sunny" Branch:

If we are working with the "Sunny" subset, we would calculate the information gain for splitting on "Temperature," "Humidity," and "Wind" within that subset and choose the attribute that yields the highest information gain to create further branches. **Outlook =**

	Temp.	Humidity	Wind	Play?
Sunny	H	H	W	-
	H	H	S	-
	M	H	W	-
	C	N	W	+
	M	N	S	+

Splitting the "Sunny" Branch

Outlook = Sunny Subset:

Temp.	Humidity	Wind	Play?
H	H	W	-
H	H	S	-
M	H	W	-
C	N	W	+
M	N	S	+

Calculating Information Gain:

- Gain(S⟨sub⟩Sunny⟨/sub⟩, Humidity) $\simeq$ 0.971 - (3/5 * 0 + 2/5 * 0) = 0.971
- Gain(S⟨sub⟩Sunny⟨/sub⟩, Temperature) = 0.571
- Gain(S⟨sub⟩Sunny⟨/sub⟩, Wind) = 0.019

Humidity is chosen for the next split.

Splitting the "Rainy" Branch

Outlook = Rainy Subset:

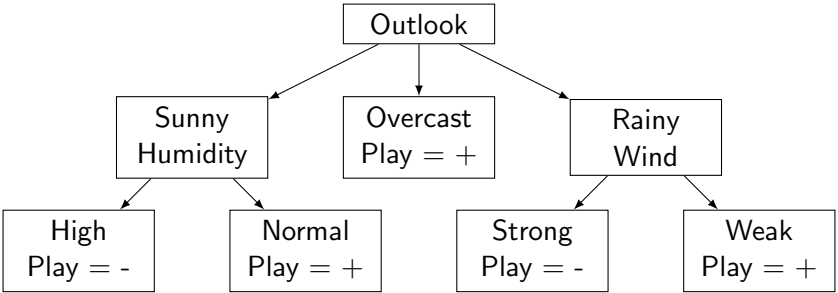
Temp.	Humidity	Wind	Play?
M	H	W	+
C	N	W	+
C	N	S	-
M	N	W	+
M	H	S	-

Calculating Information Gain:

- Gain(S⟨sub⟩Rainy⟨/sub⟩, Wind) $\simeq$ 0.971 - (3/5 * 0.918 + 2/5 * 0) = 0.419
- Gain(S⟨sub⟩Rainy⟨/sub⟩, Temperature) = 0.019
- Gain(⟨sub⟩Rainy⟨/sub⟩, Humidity) = 0.019

Wind is chosen for the next split.

The Complete Decision Tree



Python Code - Classification with DT

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/sample_data/iris.csv'
df = pd.read_csv(file_path)

# Inspect the first few rows of the dataset
print("First few rows of the dataset:")
print(df.head())

# Separate features and labels
X = df.iloc[:, :-1]  # Features: all columns except the last
y = df.iloc[:, -1]   # Labels: the last column
```

Python Code - Classification with DT

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,

# Create and train the Decision Tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Python Code - Classification with DT

```
# Visualize the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(clf, feature_names=X.columns, class_names=clf.classes_, filled=True)
plt.title("Decision Tree Visualization")
plt.show()
```

Python Code - Different Classification Algorithms

```
# Initialize classifiers
classifiers = {
    "SVM": SVC(kernel='linear', random_state=42),
    "k-NN": KNeighborsClassifier(n_neighbors=5),
    "Neural Network": MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=
}

# Train and evaluate each classifier
for name, clf in classifiers.items():
    print(f"\n=== {name} Classifier ===")
    clf.fit(X_train, y_train)  # Train the classifier
    y_pred = clf.predict(X_test)  # Make predictions
    accuracy = accuracy_score(y_test, y_pred)  # Calculate accuracy
    print(f"Accuracy: {accuracy:.2f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
```

Student Assignment: Code Comprehension and Experimentation

Objective: Deepen understanding of the machine learning workflow through code analysis and experimentation.

- **Code Dissection:** Students will meticulously analyze the provided Python code, explaining the purpose of each line and its role within the overall machine learning framework. This includes:
  - Data loading and preprocessing.
  - Model instantiation and training.
  - Prediction and evaluation.
- **Data Manipulation and Feature Engineering:** Students will explore the impact of data changes and feature engineering:
  - Experiment with different train/test splits (e.g., 80/20, 70/30, k-fold cross-validation).
  - Implement basic feature engineering techniques (e.g., feature scaling, creating interaction terms if applicable).
  - Analyze how these changes affect model performance.
- **Algorithm Comparison:** Students will extend the code to include and compare the performance of other classification algorithms from scikit-learn (e.g., Logistic Regression, Support Vector Machines, Random Forests).