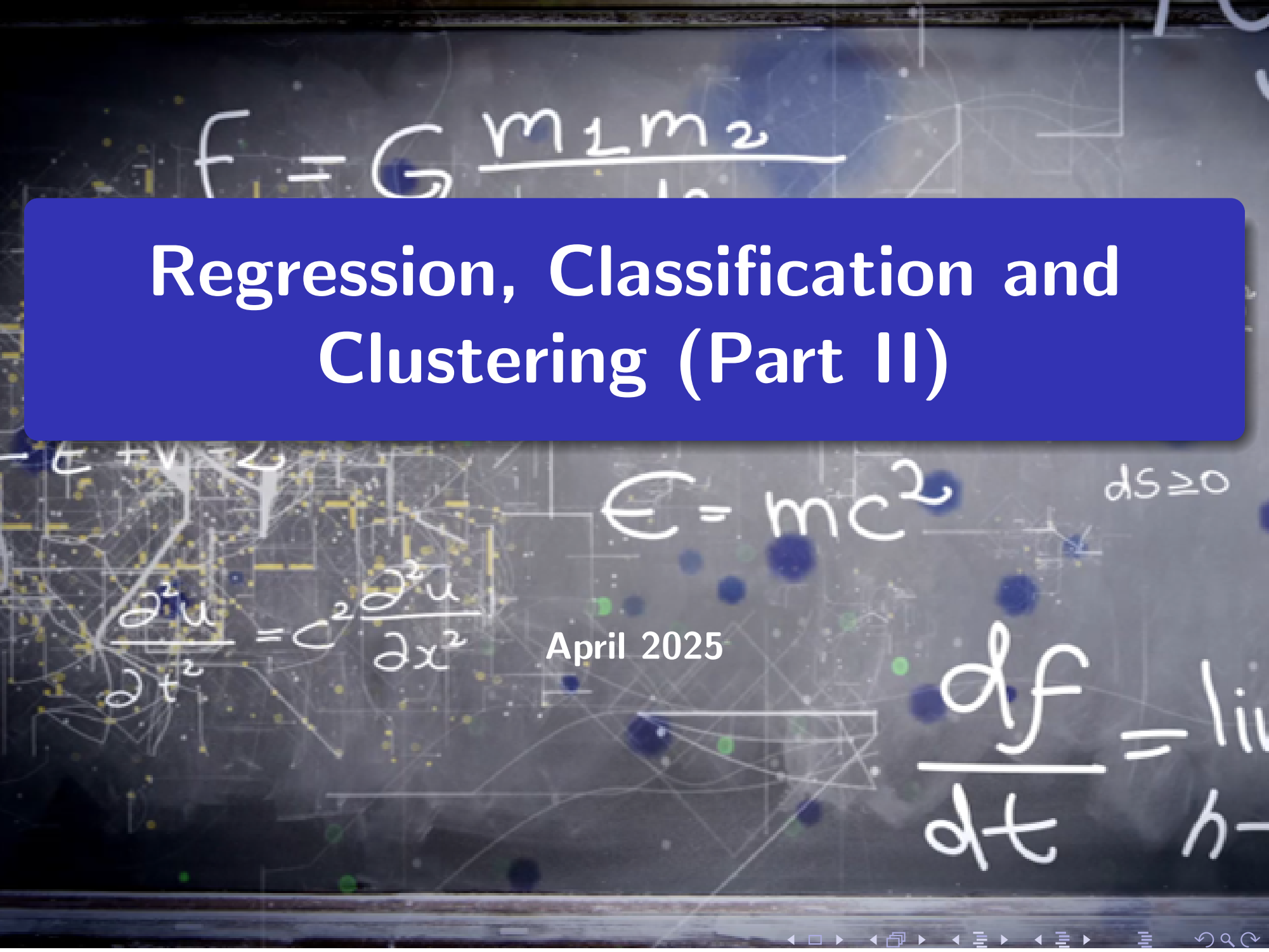


Regression, Classification and Clustering (Part II)

April 2025

Topics to be discussed

Part I. Introduction

1. Introduction to Artificial intelligence

Part II. Search and optimisation.

2. Search - basic approaches
3. Search - optimisation
4. Two-player deterministic games
5. Evolutionary and genetic algorithms

Part III. Machine learning and data analysis.

6. Regression, classification and clustering (Part I & II)
8. Artificial neural networks
9. Bayesian models
10. Reinforcement Learning

Part IV. Logic, Inference. language

11. Propositional logic and predicate logic
12. Inference

Part V. Knowledge representation, Language, Vision

13. Knowledge Representation
14. AI in Natural language processing
15. AI in Vision and Graphics

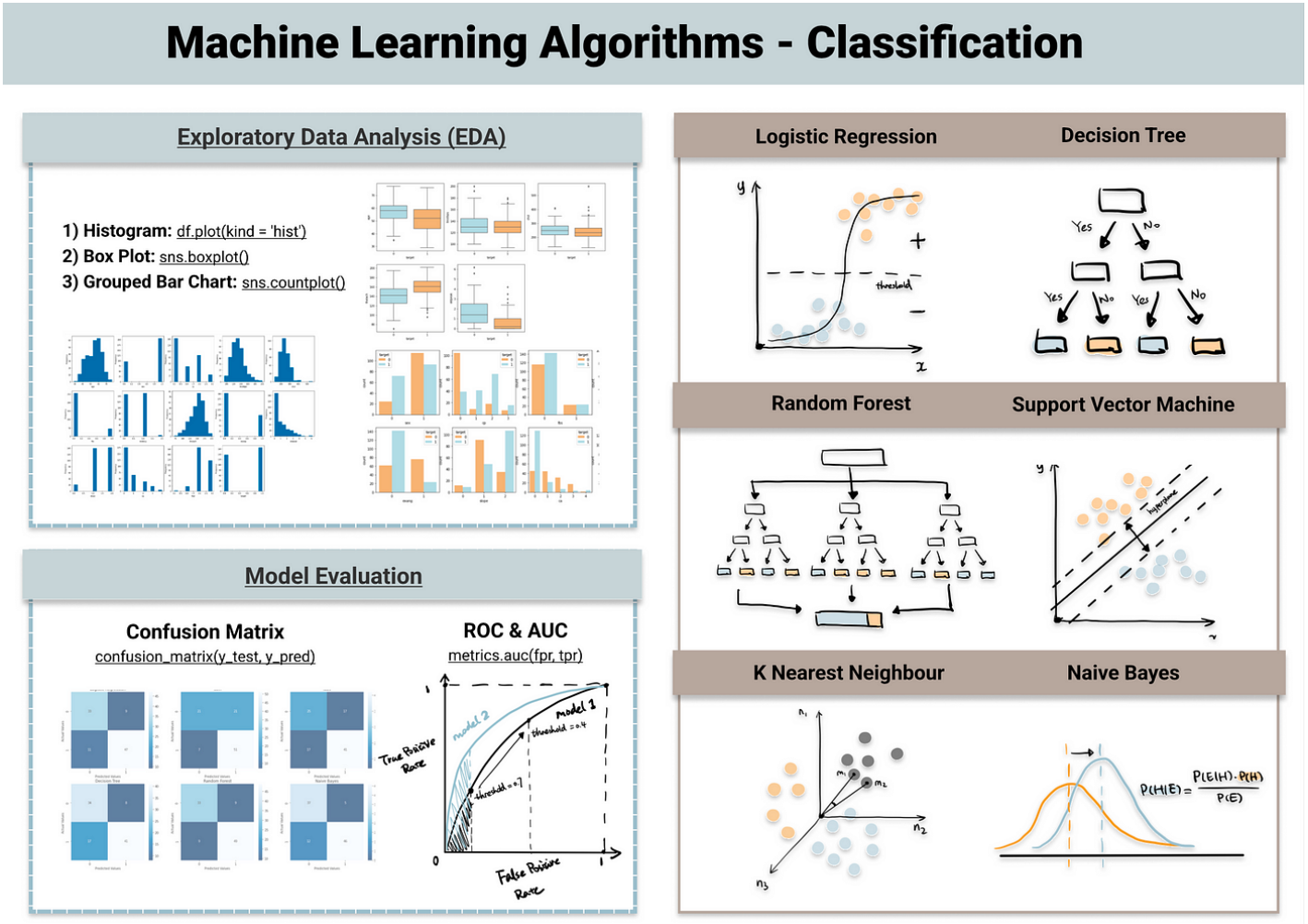
Agenda

1. Introduction
2. Hierarchical clustering
3. Associative Rule Mining
4. Support Vector Machines
5. Random Forests
6. Evaluation of Classification Models
   - Confusion matrix
   - Accuracy, precision, recall, and F1-score
   - Feature Selection and Engineering
   - Types of Feature
   - Feature Scaling and Normalization
   - Principal Component Analysis (PCA)
7. Cross Validation Methods

Previous lecture - Data Analysis (Part I) - reminder

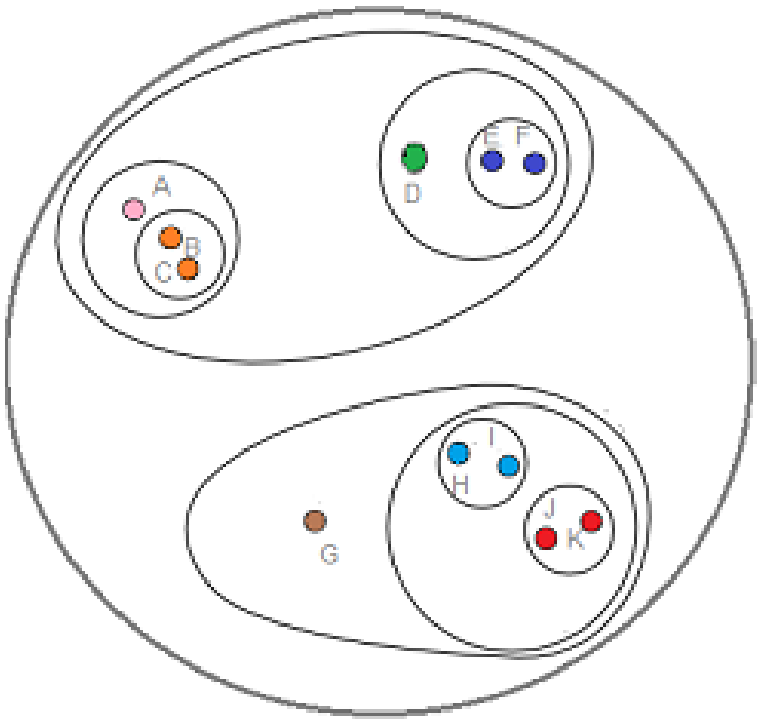
1. Overview of Analytics and Data Analysis
2. Linear Regression Analysis
3. Logistic Regression Analysis
4. Classification Analysis
5. Clustering Analysis

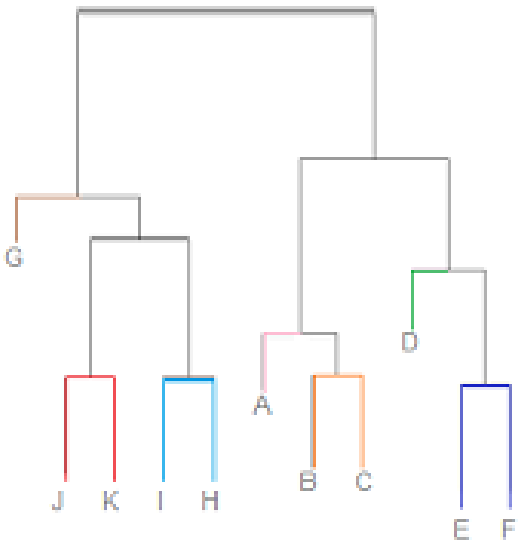
Data analysis - additional considerations



https://towardsdatascience.com/top-machine-learning-algorithms-for-classification-2197870ff501

Hierarchical Clustering





https://www.statisticshowto.com/hierarchical-clustering/

Hierarchical clustering

- **Definition**: Hierarchical clustering is a method of clustering where we build a hierarchy of clusters, which can be visualized using a dendrogram.
- **Idea**: Start with every point in a separate cluster, then iteratively merge the closest pairs of clusters, until all points belong to the same cluster. Two main methods:
  - **Agglomerative**: Start with each point in a separate cluster and successively merge the two closest clusters until all points are in the same cluster.
  - **Divisive**: Start with all points in one cluster and successively split the cluster into smaller clusters until each point is in its own cluster.
- **Distance metric**: The distance between two clusters can be defined using various metrics, such as Euclidean distance or cosine similarity.
- **Linkage criterion**: The criterion used to determine the distance between two clusters can also vary, such as single linkage (minimum distance between any two points in the two clusters), complete linkage (maximum distance), or average linkage (average distance).
- **Advantages**: No need to specify the number of clusters in advance, can handle different types of data (e.g., categorical or continuous), and can produce a visual representation of the hierarchy of clusters.
- **Disadvantages**: Computationally intensive for large datasets, sensitive to noise and outliers, and can be difficult to interpret the dendrogram for large datasets with many clusters.

Pseudocode for Agglomerative Hierarchical Clustering

Algorithm Hierarchical clustering

Input: Data matrix X, distance metric $d(\cdot, \cdot)$
Output: Dendrogram
for $i = 1$ *to* n **do**
  Create a cluster for each data point: $C_i \leftarrow \{x_i\}$
end
for $k = n$ *down to* 2 **do**
  Compute the pairwise distances between all clusters: $d_{ij} \leftarrow d(C_i, C_j)$
  Find the closest pair of clusters (C_a, C_b): $(a, b) \leftarrow \arg\min_{i,j:i<j} d_{ij}$
  Merge clusters C_a and C_b: $C_a \leftarrow C_a \cup C_b$
  Remove cluster C_b from the list of clusters
  Create a new node in the dendrogram for the merged clusters (C_a, C_b)
  for $j = 1$ *to* $k - 1$ **do**
    Update the distance between cluster C_a and cluster C_j: $d_{aj} \leftarrow \min(d_{aj}, d_{bj})$
    Remove distance d_{bj} from the list of pairwise distances
  end
end

Associative Rule Mining

- **Definition**: Associative rule mining is the process of discovering interesting relationships between variables in large datasets.
- **Idea**: The goal of associative rule mining is to identify patterns in data that occur frequently and to use those patterns to make predictions or to gain insights into the underlying structure of the data.
- **Methods**:
  - **Apriori Algorithm**: A classic algorithm that works by iteratively discovering frequent itemsets and using them to generate candidate itemsets. It prunes the search space by eliminating candidate itemsets that cannot be frequent.
  - **FP-Growth Algorithm**: A more efficient algorithm that works by building a compact data structure called an FP-tree to represent the dataset. It then recursively mines the FP-tree to generate frequent itemsets.
  - **Eclat Algorithm**: Another algorithm that works by using a depth-first search to generate frequent itemsets. It is typically faster than Apriori, but less memory-efficient.
- **Applications**:
  - Market basket analysis: Identify items that are frequently purchased together, to help with product placement, pricing, and advertising strategies.
  - Healthcare: Identify patterns in patient data to help with diagnosis, treatment, and disease prevention.
  - Web mining: Analyze web log data to discover user behavior patterns and to improve website design and user experience.

Support, Confidence, Lift

Support

The support of an itemset X is the proportion of transactions in the database that contain X.

$$\text{support}(X) = \frac{\text{number of transactions containing } X}{\text{total number of transactions}}$$

Confidence

The confidence of a rule $X \to Y$ is the proportion of transactions containing X that also contain Y.

$$\text{confidence}(X \to Y) = \frac{\text{support}(X \cup Y)}{\text{support}(X)}$$

Lift

The lift of a rule $X \to Y$ measures the degree of dependence between X and Y, and is defined as the ratio of the observed support of X and Y to the expected support if X and Y were independent.

$$\text{lift}(X \to Y) = \frac{\text{support}(X \cup Y)}{\text{support}(X) \times \text{support}(Y)}$$

Support, Confidence, Lift (cont.)

Example

Suppose we have a transactional database with 100 transactions, and the itemset $\{A, B, C\}$ appears in 20 transactions. Then the support of $\{A, B, C\}$ is:

$$\text{support}(\{A, B, C\}) = \frac{20}{100} = 0.2$$

If we also observe that $\{A, B\}$ appears in 30 transactions, then the confidence of the rule $\{A, B\} \to \{C\}$ is:

$$\text{confidence}(\{A, B\} \to \{C\}) = \frac{\text{support}(\{A, B, C\})}{\text{support}(\{A, B\})} = \frac{0.2}{0.3} \approx 0.67$$

If we further observe that the support of $\{C\}$ 0.05, then the lift of the rule $\{A, B\} \to \{C\}$ is:

$$\text{lift}(\{A, B\} \to \{C\}) = \frac{\text{support}(\{A, B, C\})}{\text{support}(A \cup B) \times \text{support}(C)} = \frac{0.2}{0.3 \times 0.05} = 13.33..$$

Apriori Algorithm

- The Apriori algorithm is an algorithm for frequent itemset mining and association rule learning over transactional databases.
- It is based on the idea that a subset of a frequent itemset must also be frequent.
- The algorithm uses a level-wise search strategy to explore the combinatorial space of itemsets, starting from small subsets of the data.

Variables used in the algorithm

- T: the set of all transactions in the database.
- I: the set of all items in the database.
- C_k: the set of candidate itemsets of size k.
- L_k: the set of frequent itemsets of size k.
- *minsup*: the minimum support threshold.

Pseudocode for Apriori Algorithm

Algorithm Apriori algorithm

Input : A database of transactions D, minimum support threshold *minsup*

Output: A set of frequent itemsets F

$L_1 \leftarrow \{f \mid f \in itemset(D)\}$ // Frequent 1-itemsets

$k \leftarrow 2$

while $L_{k-1} \neq \emptyset$ **do**

  $C_k \leftarrow \{c \mid c = i_1 \cup i_2, i_1 \in L_{k-1}, i_2 \in L_{k-1}, |i_1 \cup i_2| = k\}$ // Candidate

  foreach *transaction* $t \in D$ **do**

    $C_{t,k} \leftarrow \{c \in C_k \mid c \subseteq t\}$ // Candidate itemsets in transaction

    foreach *candidate* $c \in C_{t,k}$ **do**

      $count(c) \leftarrow count(c) + 1$ // Increment support count

  $L_k \leftarrow \{c \in C_k \mid count(c) \geq minsup\}$ // Frequent itemsets of size k

  $F \leftarrow F \cup L_k$

  $k \leftarrow k + 1$

return F

Apriori Algorithm Example

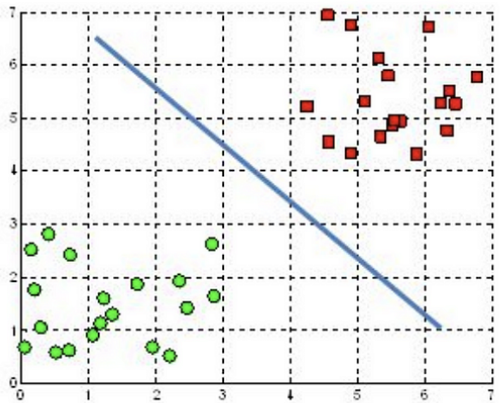
Transaction ID	Items
1	Milk, Bread, Eggs
2	Bread, Eggs, Cheese
3	Milk, Bread, Eggs, Cheese
4	Milk, Eggs
5	Bread, Eggs
6	Bread, Cheese
7	Bread, Milk, Cheese
8	Milk, Bread, Eggs
9	Milk, Bread, Cheese
10	Milk, Bread, Cheese

Algorithm Execution (no details on Support)

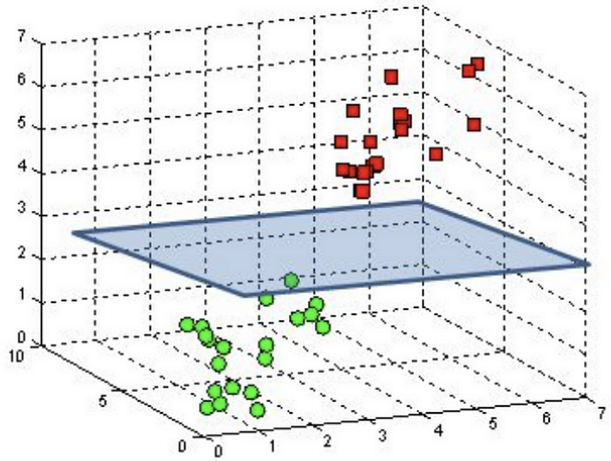
- Step 1: Find frequent 1-itemsets
  - L_1: {Milk, Bread, Eggs, Cheese}
- Step 2: Generate candidate 2-itemsets
  - C_2: {Milk, Bread}, {Milk, Eggs}, {Milk, Cheese}, {Bread, Eggs}, {Bread, Cheese}, {Eggs, Cheese}
  - Count the support of each candidate itemset in C_2
  - L_2: {Milk, Bread}, {Milk, Cheese}, {Bread, Cheese}
- Step 3: Generate candidate 3-itemsets
  - C_3: {Milk, Bread, Cheese}
  - Count the support of each candidate itemset in C_3
  - L_3: {Milk, Bread, Cheese}

Support Vector Machines (SVMs) - hyperplanes

A hyperplane in $\mathbb{R}^2$ is a line

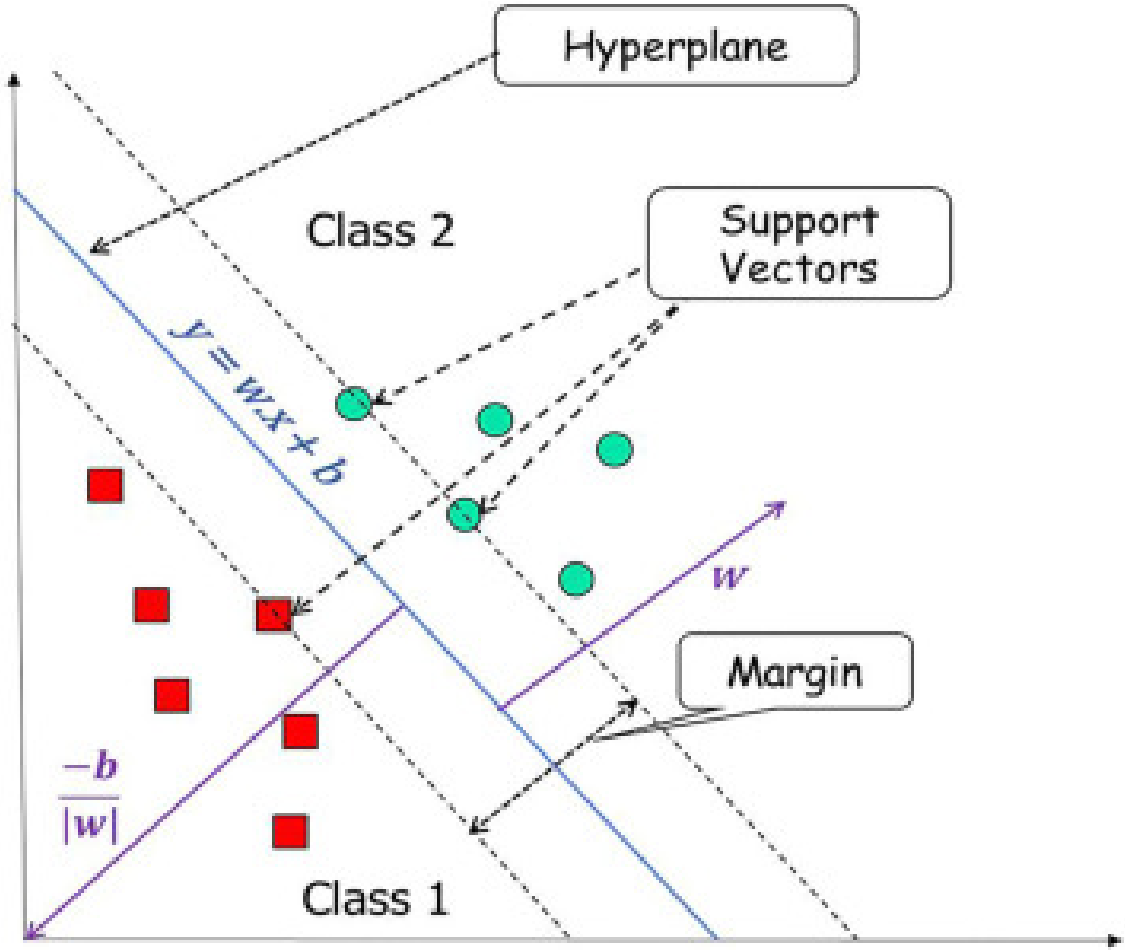


A hyperplane in $\mathbb{R}^3$ is a plane



https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

Support Vector Machines (SVMs)



https://www.sciencedirect.com/science/article/pii/B978032385214200001X

Support Vector Machines (SVMs)

Definition: Support Vector Machines (SVMs) are a type of supervised machine learning algorithm that can be used for both classification and regression tasks. The SVM algorithm finds the best hyperplane in a high-dimensional space to separate the data points into different classes.

Key Concepts:

- **Hyperplane**: A hyperplane is a decision boundary that separates the data into different classes. In two-dimensional space, a hyperplane is a line that separates the data points into different regions.
- **Margin**: The margin is the distance between the hyperplane and the closest data points from each class. SVMs try to find the hyperplane with the largest margin, as this is expected to result in better generalization performance.
- **Support Vectors**: These are the data points that are closest to the hyperplane and have the largest influence on the position of the hyperplane. In other words, these are the points that define the margin.

Support Vector Machines (SVMs) (cont.)

Math Formulas:

- The equation for a hyperplane in two-dimensional space is: $w_1 x_1 + w_2 x_2 + b = 0$, where w_1 and w_2 are the weights, x_1 and x_2 are the features, and b is the bias term.
- The equation for a hyperplane in higher-dimensional space is: $w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b = 0$, where n is the number of features.
- The distance between a data point and the hyperplane is given by: $\frac{|w^T x + b|}{\|w\|}$, where x is the data point, w is the weight vector, b is the bias term, and $\|w\|$ is the Euclidean norm of the weight vector.
- The margin is given by: $\frac{2}{\|w\|}$.
- The objective function of the SVM algorithm is to maximize the margin subject to the constraint that all data points are correctly classified. This can be formulated as the following optimization problem:

$$\text{maximize} \quad \frac{2}{\|w\|} \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1, \quad i = 1, \ldots, n$$

where y_i is the class label of the i-th data point.

SVM algorithm (stochastic gradient)

Algorithm SVM Algorithm for Classification

Input : Training data $\{(x_i, y_i)\}_{i=1}^n$, kernel function k, regularization parameter C
Output : The support vector machine classifier
Initialize: Weights w, bias term b, initial learning rate η, maximum number of iterations T
while *not converged and* $t < T$ **do**
    Randomly select a data point (x_i, y_i)
    Compute the margin: $y_i(w^T x_i + b)$
    if $y_i(w^T x_i + b) < 1$ **then**
        Update the weights and bias term:

$$w \leftarrow w + \eta(y_i x_i - 2Cw)$$
$$b \leftarrow b + \eta y_i$$

    else
        Update the weights only:

$$w \leftarrow w + \eta(-2Cw)$$

    Decrease the learning rate: $\eta \leftarrow \eta / t$
    Increment the iteration counter: $t \leftarrow t + 1$
return *The decision function:* $f(x) = sign(w^T x + b)$

Regularizations in SVM

Regularization is used in SVM to control the complexity of the model and prevent overfitting.

L1 Regularization

- The L1 norm is defined as:

$$||w||_1 = \sum_{i=1}^{n} |w_i|$$

- L1 regularization adds a penalty term proportional to the L1 norm of the weight vector to the objective function of SVM:

$$J(w) = C \sum_{i=1}^{m} \max\{0, 1 - y_i(w^T x_i + b)\} + \lambda ||w||_1$$

where λ is the regularization parameter.

L2 Regularization

- The L2 norm is defined as:

$$||w||_2 = \sqrt{\sum_{i=1}^{n} w_i^2}$$

- L2 regularization adds a penalty term proportional to the L2 norm of the weight vector to the objective function of SVM:

$$J(w) = C \sum_{i=1}^{m} \max\{0, 1 - y_i(w^T x_i + b)\} + \frac{\lambda}{2} ||w||_2^2$$

where λ is the regularization parameter.

Boosting in SVM

- Boosting is a popular ensemble learning technique that combines multiple weak classifiers to form a strong classifier.
- Boosting can be applied to SVMs by iteratively adding new support vectors to the training set and retraining the SVM with the updated training set.
- The new support vectors are selected based on their ability to improve the classification accuracy of the SVM.

Boosting Algorithm

1. Train an SVM $f_1(x)$ on the original training set D.
2. For $t = 2, 3, \ldots, T$, do:
   1. Define a new training set $D_t = \{(x_i, y_i) \mid y_i \cdot f_{t-1}(x_i) \leqslant 0\}$.
   2. Train a new SVM $f_t(x)$ on D_t.
3. Output the boosted SVM $F(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t f_t(x))$, where α_t is the weight assigned to f_t.

- The weight α_t is chosen to minimize the training error of the boosted SVM, subject to the constraint that $\alpha_t \geqslant 0$.
- Boosting can be used with various SVM regularizers, including L_1 and L_2 regularization.

Hyperparameters of SVM

- C: The cost parameter determines the trade-off between allowing misclassifications and forcing rigid margins. A smaller value of C creates a wider margin but allows more misclassifications, whereas a larger value of C creates a narrower margin but enforces stricter classification rules.
- γ: Gamma is the parameter of the Gaussian radial basis function (RBF) kernel. A small value of gamma indicates a Gaussian with a large variance, and thus the decision boundary is smoother, while a large value of gamma indicates a Gaussian with a small variance, and thus the decision boundary is more irregular.
- *kernel*: The kernel function determines the way the input data is transformed into a higher dimensional space, where it is easier to find a hyperplane that separates the classes. Common kernel functions include linear, polynomial, RBF, and sigmoid.
- *degree*: The degree of the polynomial kernel function, used only for the polynomial kernel.
- *coef*0: The independent term in the kernel function, used only for the polynomial and sigmoid kernels.
- *shrinking*: Whether to use the shrinking heuristic to speed up the training process. The shrinking heuristic discards some of the support vectors that are unlikely to affect the decision boundary, which can lead to faster training times.
- *tol*: The tolerance for stopping criterion. When the change in the objective function (i.e., the dual problem) falls below this threshold, the optimization process stops. A smaller value of tolerance leads to a more precise solution but may increase training time.

Commonly Used Kernels in SVM

- **Linear kernel:** The linear kernel is the simplest kernel and is defined as the dot product of the input features. It is given by:

$$K(x_i, x_j) = x_i^T x_j$$

- **Polynomial kernel:** The polynomial kernel maps the input features into a higher dimensional space using a polynomial function. It is given by:

$$K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$$

where γ is a hyperparameter, r is an optional coefficient, and d is the degree of the polynomial.

- **Radial basis function (RBF) kernel:** The RBF kernel maps the input features into an infinite dimensional space using a Gaussian function. It is given by:

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

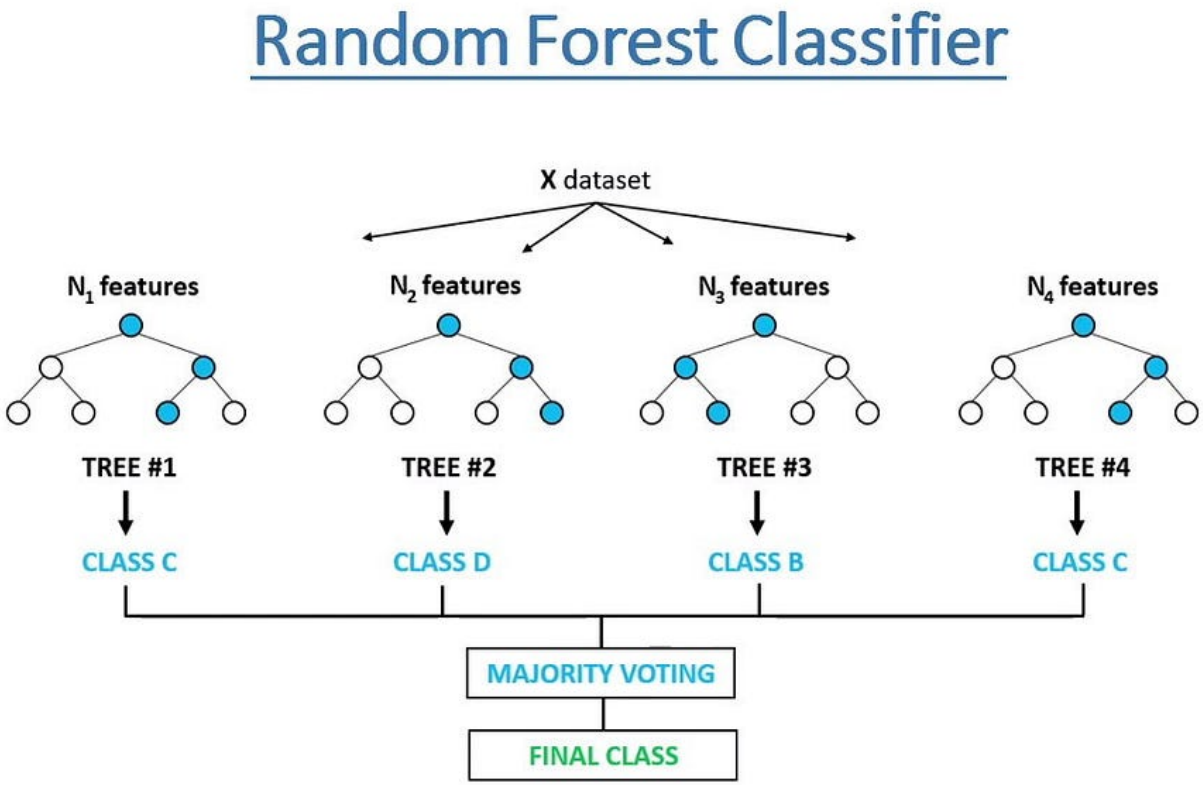
where σ is a hyperparameter that controls the width of the Gaussian.

- **Sigmoid kernel:** The sigmoid kernel maps the input features into an infinite dimensional space using a sigmoid function. It is given by:

$$K(x_i, x_j) = \tanh\left(\gamma x_i^T x_j + r\right)$$

where γ is a hyperparameter and r is an optional coefficient.

Random Forest Classifier - Introduction



https://medium.com/analytics-vidhya/random-forest-classifier-and-its-hyperparameters-8467bec755f6

Random Forests - Introduction (cont.)

Definition: Random forests are an ensemble learning method that combines multiple decision trees to improve the accuracy and robustness of the classification or regression.

How it works:

- Randomly select a subset of features from the training set.
- Build a decision tree using the selected features.
- Repeat the above steps multiple times to create a forest of decision trees.
- To classify a new instance, pass it through each decision tree in the forest and take the majority vote of the class predictions.

Advantages:

- Reduces the risk of overfitting compared to a single decision tree.
- Robust to noise and missing data.
- Can handle high-dimensional feature spaces.
- Can be used for both classification and regression tasks.

Mathematical notation:

- The decision tree is constructed using an impurity measure such as the Gini index or entropy.
- Let S be the training set and f_i be the i^{th} feature in the feature set F.
- At each node of the decision tree, the feature that maximizes the impurity reduction is selected as the splitting feature.
- The majority vote of the class predictions of all the trees is taken as the final prediction for a new instance.

Random Forests (cont.)

Mathematical notation: Given a training set $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where x_i is a feature vector and y_i is a class label, the goal is to learn a function $f : X \to Y$ that maps feature vectors to class labels.

- Each decision tree T in the forest is built using a random subset S of the training data D, where S is sampled uniformly at random from D with replacement (i.e., using bootstrap aggregation, or "bagging").
- At each node of the decision tree, a random subset of the features F is considered for splitting, where F is a subset of the full feature set X chosen randomly and independently at each node.
- The final prediction of the random forest for a new instance x is obtained by taking the majority vote of the predictions of all the individual trees in the forest, i.e.,

$$f(x) = \text{argmax}_{y \in Y} \sum_{T_i \in \text{forest}} \mathbb{I}(T_i(x) = y),$$

where $\mathbb{I}$ is the indicator function.

Evaluating Classification Models - general rules

- Use multiple evaluation metrics: Accuracy, Precision, Recall, F1-Score, AUC
- Split the data into training and testing sets to check for overfitting
- Use cross-validation to ensure the model's stability and generalizability
- Compare the model's performance against a baseline model (e.g., random guessing)
- Interpret the confusion matrix to understand the model's strengths and weaknesses
- Use different models and compare their performances to choose the best one
- Evaluate the model's performance on different datasets to ensure its applicability in different scenarios
- Continuously monitor and update the model to adapt to changing data patterns and environments

Confusion Matrix

A confusion matrix is a table used to evaluate the performance of a classification model by comparing the actual and predicted class labels of a set of test data. The matrix contains four possible outcomes:

- True Positives (TP): the number of correct positive predictions
- False Positives (FP): the number of incorrect positive predictions
- True Negatives (TN): the number of correct negative predictions
- False Negatives (FN): the number of incorrect negative predictions

The confusion matrix can be represented as follows:

	Actual Positive	Actual Negative
Predicted Positive	TP	FP
Predicted Negative	FN	TN

Accuracy, Precision, Recall, and F1-Score

The performance metrics derived from the confusion matrix include:

- **Accuracy**: The proportion of correctly classified instances out of the total number of instances:

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

  where TP, TN, FP, and FN represent the number of true positives, true negatives, false positives, and false negatives, respectively.

- **Precision**: The proportion of correctly classified positive instances out of the total number of instances predicted as positive:

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

- **Recall**: The proportion of correctly classified positive instances out of the total number of actual positive instances:

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

- **F1-Score**: The harmonic mean of precision and recall, which provides a balance between the two metrics:

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

How to Calculate TP, FP, TN, and FN - Example

- **Definition**: In a binary classification problem, TP (true positives), FP (false positives), TN (true negatives), and FN (false negatives) are used to evaluate the performance of a classification model.
- **Calculation Method**: Given a set of predictions and actual labels, TP, FP, TN, and FN can be calculated as follows:
  - True positives (TP): the number of correctly predicted positive instances
  - False positives (FP): the number of incorrectly predicted positive instances
  - True negatives (TN): the number of correctly predicted negative instances
  - False negatives (FN): the number of incorrectly predicted negative instances
- **Example**: Consider a binary classification problem where we want to classify emails as spam or not spam. We have a test set of 100 emails, and the following table shows the model's predictions and the actual labels:

Email ID	Prediction	Actual Label
1	Spam	Spam
2	Not Spam	Not Spam
3	Not Spam	Not Spam
4	Spam	Not Spam
5	Spam	Spam

Using this table, we can calculate the TP, FP, TN, and FN values as follows:

- True positives (TP) $= 2$
- False positives (FP) $= 1$
- True negatives (TN) $= 2$
- False negatives (FN) $= 1$

Examples of Using Accuracy, Precision, Recall, and F1-Score to Evaluate Classification Models

- **Example 1 - Spam Detection**: A spam detection model has been trained on a dataset of emails, with the goal of classifying each email as either spam or not spam. The model's performance is evaluated on a test set of emails, and the following results are obtained:
  - True positives (TP) = 100
  - True negatives (TN) = 800
  - False positives (FP) = 50
  - False negatives (FN) = 50

  Using these values, we can calculate the following evaluation metrics:
  - Accuracy = 0.92
  - Precision = 0.67
  - Recall = 0.67
  - F1-Score = 0.67

- **Example 2 - Medical Diagnosis**: A medical diagnosis model has been trained on a dataset of patient records, with the goal of classifying each patient as either having a certain disease or not having the disease. The model's performance is evaluated on a test set of patient records, and the following results are obtained:
  - True positives (TP) = 90
  - True negatives (TN) = 450
  - False positives (FP) = 50
  - False negatives (FN) = 10

  Using these values, we can calculate the following evaluation metrics:
  - Accuracy = 0.92
  - Precision = 0.64
  - Recall = 0.9
  - F1-Score = 0.75

True Positive Rate (TPR) and False Positive Rate (FPR)

True Positive Rate (TPR)

- Measures the proportion of actual positives correctly identified.
- Also known as sensitivity.
- Formula:

$$TPR = \frac{TP}{TP + FN}$$

False Positive Rate (FPR)

- Measures the proportion of actual negatives incorrectly identified as positives.
- Formula:

$$FPR = \frac{FP}{FP + TN}$$

ROC Curve and Area Under the Curve (AUC)

Definition:

- The Area Under the Curve (AUC) is a metric that measures the performance of a binary classification model based on the trade-off between its true positive rate (TPR) and false positive rate (FPR).
- AUC is the area under the Receiver Operating Characteristic (ROC) curve.

Description:

- The ROC curve is a graphical representation of the true positive rate (TPR) versus the false positive rate (FPR) at different classification thresholds.
- The TPR is the proportion of actual positive cases that are correctly identified as positive, while the FPR is the proportion of actual negative cases that are incorrectly identified as positive.
- AUC is the probability that a randomly selected positive instance will be ranked higher than a randomly selected negative instance by the classification model.

Interpretation:

- AUC ranges from 0 to 1, where a value of 0 indicates a poor model that always predicts the wrong class, and a value of 1 indicates a perfect model that always predicts the correct class.
- AUC of 0.5 indicates a random guessing classifier, while a value greater than 0.5 indicates that the model is better than random.

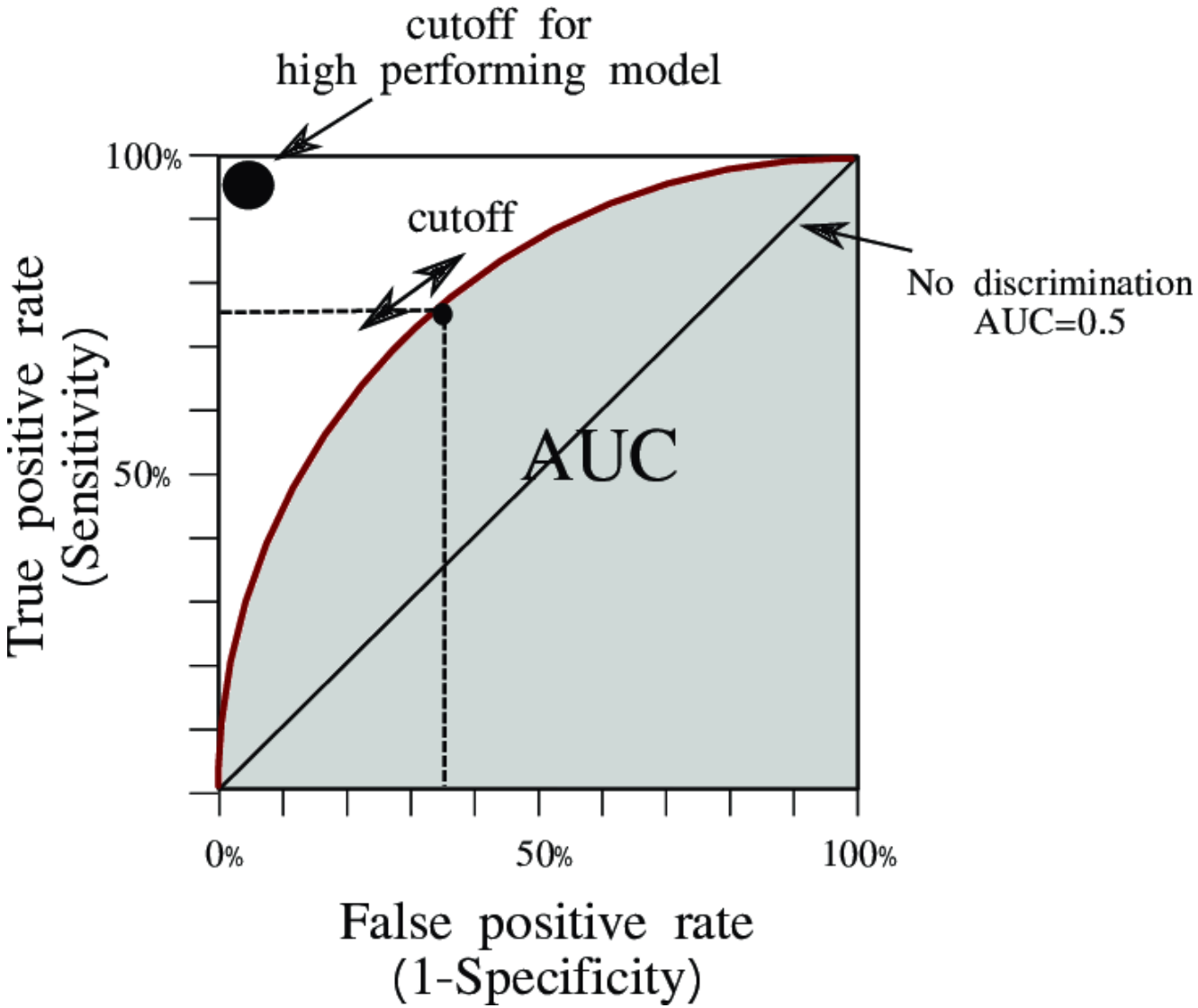
Advantages:

- AUC is a good metric for imbalanced datasets where the negative class is dominant, and it is not affected by the decision threshold of the classification model.

Disadvantages:

- AUC does not give any information about the actual performance of the model, such as accuracy, precision, or recall.

ROC-curves-and-area-under-curve-AUC



https://www.researchgate.net/publication/351506473

How to Calculate AUC

The AUC can be calculated using the following steps:

1. Sort the predicted probabilities in descending order
2. Set the threshold to be below the minimum probability, so that all observations are classified as positive
3. Move the threshold up to the next highest predicted probability, and calculate the true positive rate (TPR) and false positive rate (FPR) at that threshold
4. Continue moving the threshold up and calculating TPR and FPR until you reach the maximum predicted probability, where all observations are classified as negative
5. Calculate the area under the ROC curve using the trapezoidal rule:

$$AUC = \int_0^1 TPR(FPR^{-1}(t))dt \approx \sum_{i=1}^{n-1} \frac{(FPR_i - FPR_{i-1})(TPR_i + TPR_{i-1})}{2}$$

where:

- $FPR^{-1}(t)$ is the inverse function of the FPR with respect to the threshold t
- FPR_i and TPR_i are the FPR and TPR at the i-th threshold
- n is the total number of thresholds.

Examples of using AUC to evaluate classification models

The AUC metric can be used to evaluate the performance of a binary classification model. Here are some examples of how it can be used:

- **Comparing models:** AUC can be used to compare the performance of two or more classification models on the same dataset. The model with the higher AUC is generally considered to be the better one.
- **Tuning hyperparameters:** AUC can be used as a metric to tune hyperparameters of a classification model. For example, if you are using a logistic regression model, you can vary the regularization parameter and choose the value that maximizes the AUC.
- **Imbalanced datasets:** AUC can be particularly useful when evaluating models on imbalanced datasets. In such datasets, the majority class may dominate the evaluation metric such as accuracy, but AUC is sensitive to the performance of the minority class as well.
- **Threshold selection:** AUC can also be used to select the optimal threshold for a binary classification model. The threshold is the value that determines whether a predicted probability belongs to the positive or negative class. By choosing the threshold that maximizes the AUC, we can achieve a balance between sensitivity and specificity.

Definition of Feature Selection and Engineering

Feature Selection

- The process of selecting a subset of relevant features for use in model construction.
- Reduces overfitting, improves accuracy, and reduces training time.
- Common methods:
  - Wrapper methods (e.g., Recursive Feature Elimination, Forward/Backward Stepwise Selection)
  - Filter methods (e.g., Correlation-based Feature Selection, Mutual Information-based Feature Selection)
  - Embedded methods (e.g., Lasso, Ridge Regression)

Feature Engineering

- The process of creating new features from the existing ones to improve model performance.
- Requires domain knowledge and creativity.
- Common methods:
  - Scaling/Normalization (e.g., Min-Max scaling, Z-score normalization)
  - Imputation (e.g., mean imputation, k-Nearest Neighbors imputation)
  - Transformation (e.g., log transformation, PCA)
  - Creation of new features (e.g., interaction features, polynomial features)

Feature Types

- Numerical Features - features with continuous numerical values.
  - Normal Distribution - A normal distribution, also known as a Gaussian distribution, is a probability distribution that is symmetrical and bell-shaped. Many numerical features follow a normal distribution, and this can be useful for selecting appropriate machine learning algorithms and for data preprocessing.
  - Outliers - Outliers are data points that are significantly different from other data points in the dataset. Outliers can skew statistical analyses and machine learning algorithms.
- Categorical Features - features with discrete values that represent categories
  - One-Hot Encoding - One-hot encoding is a technique used to represent categorical data in a format. Each category is represented by a binary feature, where the presence of a category is denoted by a value of 1 and the absence is denoted by a value of 0.
  - Label Encoding - Label encoding is another technique used to represent categorical data in a numerical format. In label encoding, each category is assigned a unique integer value. However, this can introduce an arbitrary ordering to the categories that may not be appropriate for all machine learning algorithms.
- Textual Features - features that represent text data (ex: film descriptions, user reviews).
  - Bag of Words - Bag of words is a technique used to convert textual data into a numerical format. It involves creating a vocabulary of all unique words in the dataset and then representing each text document as a vector of word counts. This technique does not consider the order or context of words in the document.
  - TF-IDF - Term Frequency-Inverse Document Frequency (TF-IDF) is another technique used to convert textual data into a numerical format. It takes into account the frequency of words in a document as well as the frequency of words across all documents in the dataset. This can help to identify important words that are unique to specific documents.
- Image Features - These are features that represent image data like: pixel values, color histograms, and texture features.
- Audio Features - These are features that represent audio data like: waveform, frequency spectrum, and spectrogram.

Feature Scaling and Normalization

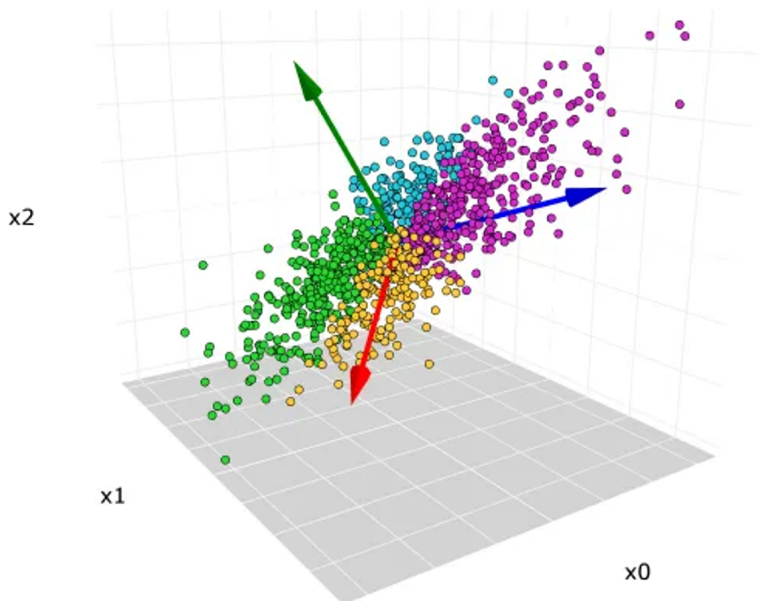
- Definition:
  - Feature Scaling: Scaling the values of the features to be within a specific range.
  - Feature Normalization: Scaling the values of the features to have zero mean and unit variance.
- Methods:
  - Min-Max Scaling: Scaling the values to be within a specific range (e.g., [0, 1]).
  - Standardization: Scaling the values to have zero mean and unit variance, using the formula:

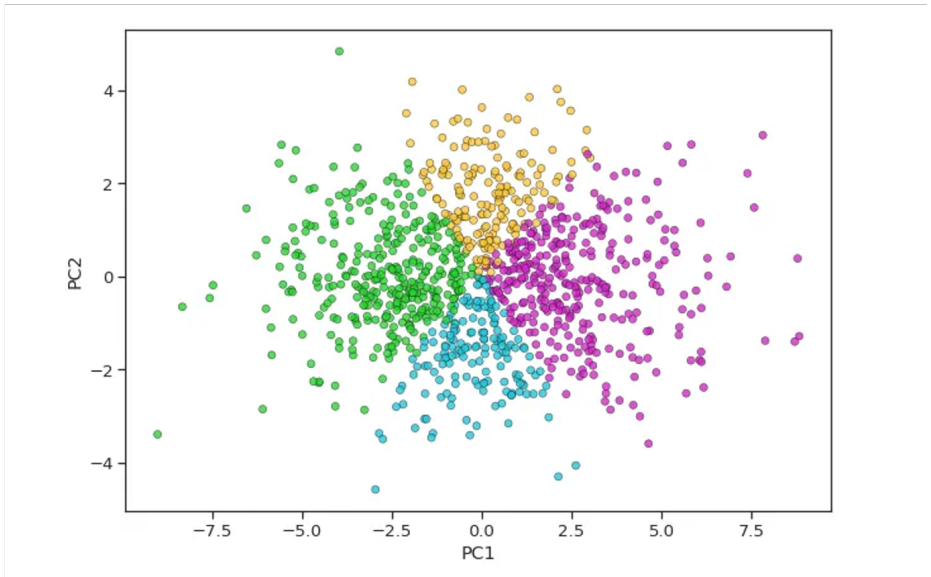
$$\widetilde{X} = \frac{X - \mu}{\sigma}$$

    where $\widetilde{X}$ is the standardized feature, X is the original feature, μ is the mean of the feature values, and σ is the standard deviation of the feature values.
  - Robust Scaling: Scaling the values to be within a specific range, using the median and interquartile range (IQR) instead of the mean and standard deviation, to be less sensitive to outliers.
- Examples:
  - Min-Max Scaling: Used in image processing to scale the pixel values to [0, 1].
  - Standardization: Used in linear regression to ensure that all features are on the same scale.
  - Robust Scaling: Used in financial analysis to scale stock prices, which may have outliers.

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a linear transformation technique that is used to reduce the dimensionality of a dataset while preserving its important features.





https://towardsdatascience.com/principal-component-analysis-pca-explained-visually-with-zero-math-1cbf392b9e7d

Principal Component Analysis (PCA) (cont.)

- Definition: PCA is a dimensionality reduction technique that finds the directions of maximum variance in a dataset and projects the data onto a lower-dimensional subspace spanned by these directions.
- Methods:
  - Standardize the data (subtract mean, divide by standard deviation)
  - Compute the covariance matrix $\mathbf{\Sigma}$ of the standardized data
  - Compute the eigenvectors and eigenvalues of $\mathbf{\Sigma}$ using eigendecomposition
  - Sort the eigenvectors in decreasing order of eigenvalues to obtain the principal components (PCs)
  - Project the data onto the PCs to obtain the lower-dimensional representation
- Examples:
  - Image compression: reduce the number of pixels in an image while preserving the essential features by projecting the pixel values onto the PCs
  - Genetics: identify the most important genetic markers for a disease by projecting the genetic data onto the PCs
  - Financial analysis: reduce the dimensionality of financial data (e.g. stock prices) to identify patterns and trends

PCA Pseudocode

Algorithm PCA Algorithm

Data: Data matrix X, Number of principal components k
Result: Projected data matrix Y

- Center the data: $\widetilde{X} \leftarrow X - \frac{1}{n}\sum_{i=1}^{n} X_i$
- Calculate the covariance matrix: $C \leftarrow \frac{1}{n-1}\widetilde{X}^T\widetilde{X}$
- Calculate the eigenvectors and eigenvalues of C: $(\lambda_i, \mathbf{v}_i) \leftarrow \text{eig}(C)$
- Sort eigenvectors in descending order of eigenvalues: $\mathbf{V} \leftarrow [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d]$
- Choose the first k eigenvectors: $\mathbf{V}_k \leftarrow [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k]$
- Project the data onto the new subspace: $Y \leftarrow \widetilde{X}\mathbf{V}_k$

where:

- $\widetilde{X}$ is the projected matrix
- X_c is the centered data matrix
- C is the sample covariance matrix
- W is the projection matrix

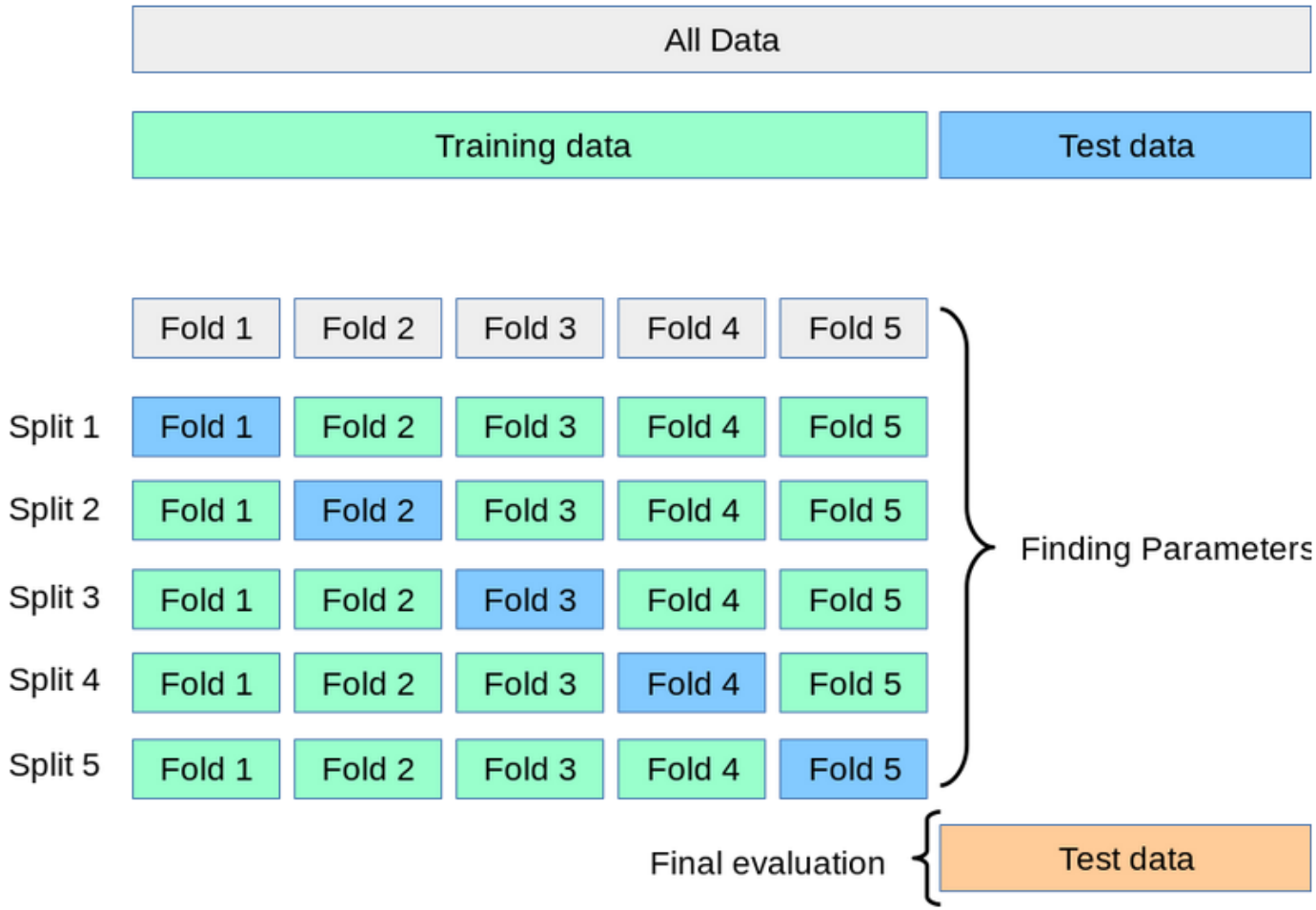
Cross Validation Methods

- **Cross validation** is a technique used to evaluate the performance of a machine learning model.
- It involves partitioning the data into several subsets, or "folds", and then training and evaluating the model on each fold.
- The performance of the model is then averaged across all folds to obtain an estimate of its generalization performance.

Common Cross Validation Methods

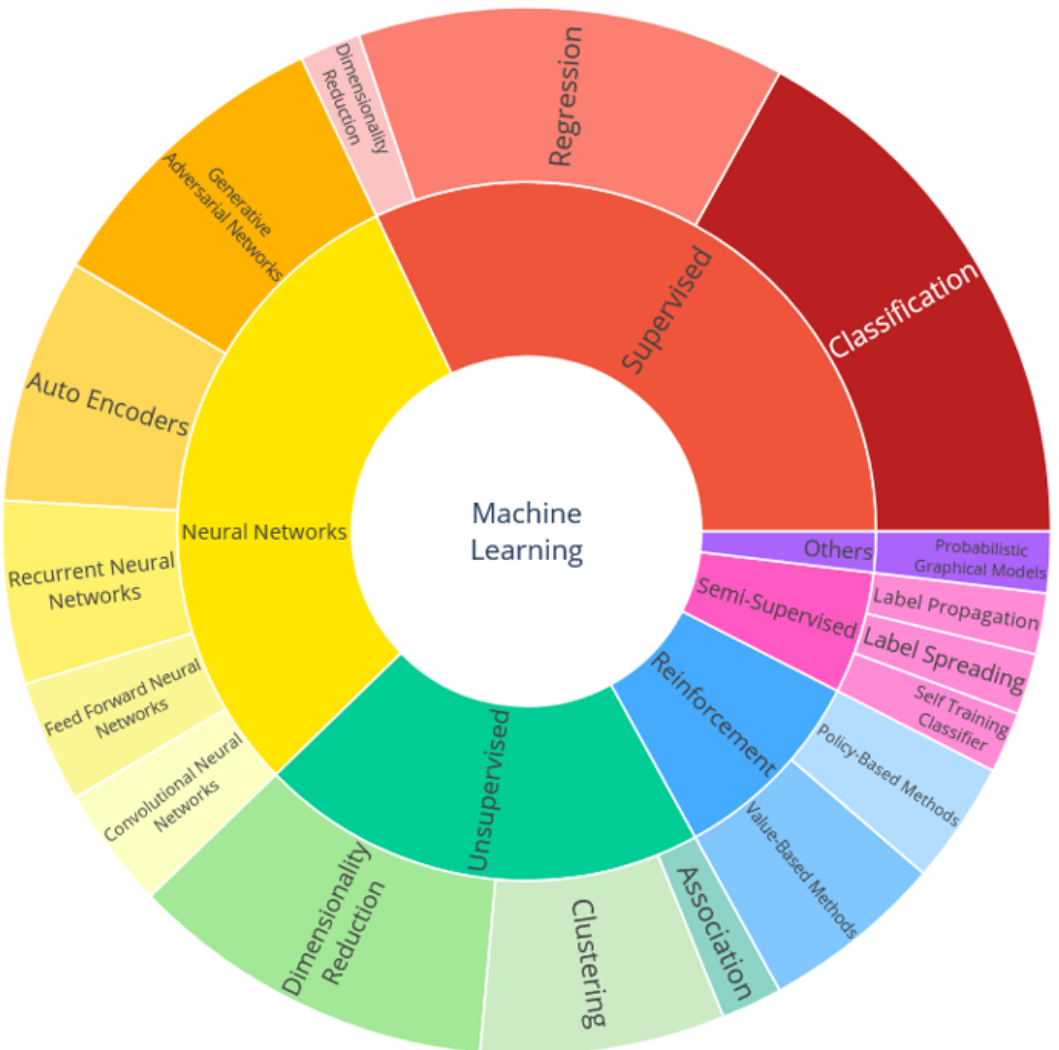
- **k-Fold Cross Validation**: The data is divided into k equally sized folds, and the model is trained and evaluated k times, each time using a different fold as the validation set and the remaining folds as the training set.
- **Leave-One-Out Cross Validation (LOOCV)**: Each data point is used as a validation set in turn, and the model is trained on the remaining data points.
- **Stratified Cross Validation**: Used when the data is imbalanced. It ensures that each fold has a similar distribution of classes as the full dataset.
- **Time Series Cross Validation**: Used for time series data where the order of the data points matters. The data is divided into folds based on time periods, and the model is trained on past data and evaluated on future data.

Cross Validation Methods (cont.)



https://scikit-learn.org/stable/modules/cross_validation.html

ML and its algorithms - summary



https://towardsdatascience.com/apriori-algorithm-for-association-rule-learning-how-to-find-clear-links-between-transactions-bf7ebc22cf0a

Summary - Regression, Classification, Clustering (Part II)

1. Introduction
2. Hierarchical clustering
3. Associative Rule Mining
4. Support Vector Machines
5. Random Forests
6. Evaluation of Classification Models
   - Confusion matrix
   - Accuracy, precision, recall, and F1-score
   - Feature Selection and Engineering
   - Types of Feature
   - Feature Scaling and Normalization
   - Principal Component Analysis (PCA)
7. Cross Validation Methods

References

1. S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach", Financial Times Prentice Hall, 2019.
2. T. Nield, "Essential Math for Data Science: Take Control of Your Data with Fundamental Linear Algebra, Probability, and Statistics", O'Reilly Media, 2022
3. A. Geron, "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow", O'Reilly Media, 3rd ed., 2023
4. M. Flasiński, Introduction to Artificial Intelligence", Springer Verlang, 2016
5. M. Muraszkiewicz, R. Nowak (ed.), Sztuczna Inteligencja dla inżynierów", Oficyna Wydawnicza PW, 2022
6. J. Prateek , Artificial Intelligence with Python", Packt 2017