Linear Regression

Linear regression models the relationship between input features and a continuous target variable by fitting a linear equation:

$$\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b \tag{1}$$

Training Procedure (Gradient Descent)

- **1. Initialize weights:** Start with random values for w_i and b.
- **2. Make predictions:** $\hat{y} = Xw + b$
- **3. Compute loss:** Mean Squared Error (MSE)

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

- **4. Compute gradients:**

$$\frac{\partial \text{Loss}}{\partial w_j} = \frac{2}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_{ij} \tag{3}$$

$$\frac{\partial \text{Loss}}{\partial b} = \frac{2}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \tag{4}$$

- **5. Update weights:**

$$w_j = w_j - \alpha \cdot \frac{\partial \text{Loss}}{\partial w_j}, \quad b = b - \alpha \cdot \frac{\partial \text{Loss}}{\partial b} \tag{5}$$

- **6. Repeat:** Iterate until convergence.

Assumptions of Linear Regression

1. Linearity

Explanation: The relationship between features and the target is linear.
Check: Plot residuals vs. predicted values — should show no clear pattern.
Fix: Add polynomial or interaction terms, apply feature transformations, or switch to non-linear models.

2. Independence of Errors

Explanation: Residuals should be independent across observations.
Check: Use the Durbin-Watson test or plot residuals over time (for time series).
Fix: Use time series models like ARIMA or include lag variables.

3. Homoscedasticity

Explanation: The variance of residuals should be constant across all levels of the predicted values.
Check: Plot residuals vs. fitted values — look for ”funnel” shapes.
Fix: Apply a transformation to the target (e.g., log), or use weighted least squares.

4. No Multicollinearity

Explanation: Features should not be highly correlated with each other.
Check: Calculate the Variance Inflation Factor (VIF) — values ¿ 5 or 10 are problematic.
Fix: Remove or combine correlated features, use PCA, or apply regularization (Lasso/Ridge).

5. Normality of Errors

Explanation: Residuals should be normally distributed for valid inference (e.g., confidence intervals).
Check: Q-Q plot or histogram of residuals.
Fix: Transform the target variable (e.g., log, Box-Cox), or use robust regression techniques.

Logistic Regression

Logistic regression is a classification algorithm used to model the probability that a binary target variable y belongs to class 1:

$$P(y = 1|x) = \hat{y} = \sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}} \tag{6}$$

where $\sigma(\cdot)$ is the sigmoid function.

Training Procedure (Gradient Descent)

- **1. Initialize weights:** Random values for w and b.
- **2. Make predictions:**
$$\hat{y}_i = \frac{1}{1 + e^{-(w^T x_i + b)}} \tag{7}$$
- **3. Compute loss:** Binary Cross-Entropy
$$\text{Loss} = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{8}$$
- **4. Compute gradients:**
$$\frac{\partial \text{Loss}}{\partial w_j} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) x_{ij}, \quad \frac{\partial \text{Loss}}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \tag{9}$$
- **5. Update weights:**
$$w_j = w_j - \alpha \cdot \frac{\partial \text{Loss}}{\partial w_j}, \quad b = b - \alpha \cdot \frac{\partial \text{Loss}}{\partial b} \tag{10}$$
- **6. Repeat:** Until convergence.

Assumptions of Logistic Regression

1. Linearity of Log-Odds

Explanation: The log-odds (not the probability) must be linearly related to the features.
Check: Plot logit($\hat{y}$) vs. each x or use Box-Tidwell test.
Fix: Add interaction or polynomial terms, transform variables, or use non-linear models.

2. Independence of Observations

Explanation: Each observation must be independent from others.
Check: Review data collection process or use time-series plots if data is ordered.
Fix: Use time-series or mixed-effects models for dependent data.

3. No Multicollinearity

Explanation: Predictors should not be highly correlated.
Check: Compute Variance Inflation Factor (VIF).
Fix: Remove/merge correlated variables, apply PCA, or use regularization.

4. Large Sample Size

Explanation: Logistic regression performs best with many observations.
Check: Rule of thumb: at least 10 events per predictor.
Fix: Collect more data or reduce the number of predictors.

5. No Extreme Outliers

Explanation: Outliers can distort the model coefficients.
Check: Use boxplots, leverage, and Cook's distance.
Fix: Remove or transform outliers, or use robust logistic regression.

Integrating Regularization with Regression Models

1. Base Models

1.1 Linear Regression (Continuous Target)

$$\text{Loss}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{11}$$

Used when: Predicting continuous numerical values.

1.2 Logistic Regression (Binary Classification)

$$\text{Loss}_{\text{log}} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] \tag{12}$$

Used when: Predicting binary outcomes ($y_i \in \{0, 1\}$).

2. Regularization Techniques

2.1 Ridge Regression (L2 Regularization)

Linear Regression:

$$\text{Loss}_{\text{ridge}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{p} w_j^2 \tag{13}$$

Logistic Regression:

$$\text{Loss}_{\text{ridge-log}} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] + \lambda\sum_{j=1}^{p} w_j^2 \tag{14}$$

Explanation: Shrinks large weights, useful when many features are relevant. Does not force coefficients to zero.
When to use: Use when overfitting is a concern and all features might contribute.

2.2 Lasso Regression (L1 Regularization)

Linear Regression:

$$\text{Loss}_{\text{lasso}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{p} |w_j| \tag{15}$$

Logistic Regression:

$$\text{Loss}_{\text{lasso-log}} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] + \lambda\sum_{j=1}^{p} |w_j| \tag{16}$$

Explanation: Encourages sparsity, setting some weights to exactly zero. Helps in feature selection.
When to use: Use when you suspect only a few features are important.

2.3 Elastic Net (Combination of L1 and L2)

Linear Regression:

$$\text{Loss}_{\text{elastic}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^{p} |w_j| + \lambda_2 \sum_{j=1}^{p} w_j^2 \tag{17}$$

Logistic Regression:

$$\text{Loss}_{\text{elastic-log}} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] + \lambda_1 \sum_{j=1}^{p} |w_j| + \lambda_2 \sum_{j=1}^{p} w_j^2 \tag{18}$$

Explanation: Combines benefits of Ridge and Lasso. Encourages both small and sparse coefficients. Useful when there are many correlated features.

When to use: Use when features are correlated and you need both regularization and feature selection.

3. Comparison Table

Model	Loss Function	Effect	When to Use
Linear Regression	MSE	No regularization	Predict continuous target
Logistic Regression	Log loss	No regularization	Binary classification
Ridge (L2)	MSE or Log loss + $\lambda \sum w_j^2$	Shrinks weights	Many features are useful
Lasso (L1)	MSE or Log loss + $\lambda \sum \lvert w_j \rvert$	Sparse model	Few features are important
Elastic Net	MSE or Log loss + both penalties	Mix of shrinkage and sparsity	Correlated + sparse features

Multinomial Logistic Regression (Softmax Regression)

Multinomial Logistic Regression is a generalization of binary logistic regression to multi-class problems. It models the probabilities of each class using the **softmax** function and is suitable when the target variable has more than two categories.

Problem Setup

Given a dataset with input features $X \in \mathbb{R}^{n \times d}$ and target labels $y \in \{1, 2, \ldots, K\}$ for K classes, the goal is to estimate the probability that a given input belongs to each class.

Model

Each class k has an associated weight vector $\mathbf{w}_k$ and bias b_k. The linear score for class k is:

$$z_k = \mathbf{w}_k^\top \mathbf{x} + b_k$$

Softmax Function

The softmax function converts the raw scores into probabilities:

$$P(y = k \mid \mathbf{x}) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

Loss Function (Cross-Entropy)

The loss is the negative log-likelihood of the true class:

$$\mathcal{L} = -\sum_{i=1}^{n}\sum_{k=1}^{K} \nVdash(y_i = k) \log P(y_i = k \mid \mathbf{x}_i)$$

Optimization

The parameters $\{\mathbf{w}_k, b_k\}_{k=1}^{K}$ are optimized using gradient descent or its variants.

When to Use

- When the target variable has more than two classes (multi-class classification).
- When classes are mutually exclusive.
- Preferred when probabilistic interpretation is desired.

Advantages

- Provides class probabilities.
- Extends binary logistic regression naturally.
- Works well for linearly separable classes.

Limitations

- Assumes independence between classes (no correlation).
- May struggle with non-linear decision boundaries (use kernel methods or other models).

Decision Tree

A Decision Tree is a supervised learning algorithm used for both classification and regression. It recursively splits the dataset into smaller subsets based on feature values, forming a tree where internal nodes are decision rules and leaves represent outcomes.

Splitting Procedure (General)

Goal

To find the best feature and threshold at each node that results in the purest possible child nodes, according to a defined impurity criterion.

Steps

1. For each feature j:
   (a) Sort the unique values of the feature.
   (b) Generate candidate split thresholds between each pair of consecutive values:
   $$\text{Threshold} = \frac{x_i + x_{i+1}}{2}$$
   (c) For each threshold t, split the dataset into:
      - Left: $x_j \leq t$
      - Right: $x_j > t$

   (d) Compute impurity of the split using a suitable criterion (Gini, Entropy, or MSE).
2. Choose the feature and threshold that yields the highest impurity reduction (i.e., best split).

Classification Tree Procedure

Impurity Measures

- **Gini Impurity:**

$$Gini = 1 - \sum_{i=1}^{C} p_i^2$$

- **Entropy:**

$$Entropy = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

- **Information Gain:**

$$IG = Impurity_{\text{parent}} - \sum_{k=1}^{K} \frac{n_k}{n} \cdot Impurity_{\text{child}_k}$$

Stopping Conditions

- Maximum depth reached
- Node has fewer than a minimum number of samples
- All samples belong to one class
- No split improves impurity

Prediction

For a new sample, follow the decision path from root to a leaf. Output the majority class in the leaf.

Regression Tree Procedure

Impurity Measure

- **Mean Squared Error (MSE):**

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

- **Variance Reduction:**

$$\text{Reduction} = Var_{\text{parent}} - \sum_{k=1}^{K} \frac{n_k}{n} \cdot Var_{\text{child}_k}$$

Stopping Conditions

- Maximum tree depth
- Minimum number of samples in a node
- Minimum MSE reduction threshold not met

Prediction

Traverse the tree using the feature values of the test sample until reaching a leaf. Return the mean target value of the samples in that leaf:

$$\hat{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

Handling Missing Values

Training Phase

- **Surrogate Splits:** When the best split feature has missing values, find surrogate features that result in similar splits and use them instead.
- **Distribution-based Assignment:** Assign samples with missing values to both left and right child nodes based on the proportion of known samples.
- **Imputation:** Preprocess the data by imputing missing values using strategies like mean, median, or mode.

Prediction Phase

- **Use of Surrogate Splits:** If a feature value is missing, the tree uses surrogate splits (backup features) to decide the direction.
- **Soft Traversal:** Allow the instance to traverse both branches with weights proportional to observed training distributions, then aggregate the results.

Notes

- Decision trees do not require feature scaling or normalization.
- They can model complex, non-linear relationships.
- Trees are prone to overfitting and can be regularized using pruning or hyperparameter tuning.

Pruning Techniques in Decision Trees

Pruning is used to reduce overfitting in decision trees by simplifying the tree structure. It limits the tree's complexity and improves generalization.

1. Pre-pruning (Early Stopping)

Pre-pruning halts the tree growth early by applying certain conditions during training. It prevents the model from growing branches that may lead to overfitting.

Common pre-pruning strategies:

- `max_depth`: Limit the maximum depth of the tree.
- `min_samples_split`: Minimum number of samples required to split a node.
- `min_samples_leaf`: Minimum number of samples required to be at a leaf node.
- `max_leaf_nodes`: Limit the total number of leaf nodes in the tree.

Advantages:

- Reduces training time.
- Controls model complexity and helps generalize better.

2. Post-pruning (Cost Complexity Pruning)

Post-pruning involves growing a full tree first, and then pruning back branches that have little importance using a validation set or complexity parameter.

Steps:

1. Grow the complete tree.
2. Evaluate subtrees based on a pruning metric (e.g., validation error, cost complexity).

3. Prune branches that don't improve performance.

   Common method: Cost complexity pruning (used in CART):

$$R_\alpha(T) = R(T) + \alpha \cdot |T|$$

Where:

- $R(T)$ is the error of the tree.
- $|T|$ is the number of terminal nodes (leaves).
- α is a complexity parameter controlling the penalty for model size.

Pruning in Random Forest

- **Pre-pruning** is commonly used (e.g., `max_depth`, `min_samples_leaf`) to control overfitting of individual trees.
- **Post-pruning** is rarely used in Random Forest because the ensemble effect already reduces overfitting.
- Fully grown (unpruned) trees are typically used to achieve low bias.

Random Forest

Random Forest is an ensemble learning method that builds multiple decision trees and combines their outputs to improve generalization and reduce overfitting. It supports both classification and regression tasks.

Key Concepts

- **Bagging (Bootstrap Aggregation):** Multiple decision trees are trained on different bootstrap samples (samples drawn with replacement) from the training data. This helps reduce variance.
- **Feature Randomness:** At each node split during tree construction, a random subset of features is selected. This reduces correlation among the trees and improves generalization.
- **Ensemble Prediction:**
  - For classification: majority vote from all trees.
  - For regression: average of all tree outputs.
- **Variance Reduction:** By averaging multiple uncorrelated trees, random forests significantly reduce model variance without increasing bias.
- **Robustness:** Random Forest is robust to noise, overfitting, and missing values. It can handle both numerical and categorical features.

Training Procedure (General)

1. For B iterations (number of trees):
   (a) Draw a bootstrap sample from the original training dataset (same size as original but sampled with replacement).
   (b) Train a decision tree on the sampled data:
      - At each node split, select a random subset of m features from total M features ($m \ll M$).
      - Among these m features, find the one with the best split based on impurity (e.g., Gini, entropy, MSE).
      - Repeat the process recursively to grow the tree.
      - Optionally apply early stopping (e.g., maximum depth, minimum samples).
2. Store the trained decision trees for ensemble prediction.

Splitting in Individual Trees

- The internal splitting logic is the same as standard decision trees, except for feature randomness.
- At each decision node:
  - Randomly select m features from the M total features.
  - For each selected feature, find the threshold that maximizes impurity reduction.
  - Select the feature-threshold pair that results in the best split.
- Common splitting criteria:
  - **Classification:** Gini impurity, entropy
  - **Regression:** Mean Squared Error (MSE)

Classification Procedure

1. Each decision tree is used to predict the class of the test sample.
2. Final prediction is determined by taking the majority vote:
$$\hat{y} = \text{mode}(y_1, y_2, \ldots, y_B)$$
3. This voting mechanism ensures robustness and reduces overfitting.

Regression Procedure

1. Each decision tree outputs a numeric prediction for the input sample.
2. The final prediction is the average of all predictions:
$$\hat{y} = \frac{1}{B}\sum_{i=1}^{B} y_i$$
3. Averaging reduces prediction variance and smooths the output.

Feature Importance

Random Forest can estimate feature importance based on:

- **Mean Decrease in Impurity (MDI):** Features that result in larger reductions of impurity are considered more important.
- **Permutation Importance:** Measure the drop in accuracy when a feature's values are randomly shuffled.

Regularization and Hyperparameters

- `n_estimators` — number of trees in the forest
- `max_features` — number of features to consider at each split (controls randomness)
- `max_depth` — maximum depth of each tree
- `min_samples_split` — minimum samples required to split a node
- `min_samples_leaf` — minimum samples required at a leaf node
- `bootstrap` — whether bootstrap samples are used when building trees

Out-of-Bag (OOB) Error Estimation

Out-of-Bag (OOB) error is an internal validation method used in Random Forests to estimate generalization error without the need for a separate validation set or cross-validation.

Key Idea

Each tree in the forest is trained on a bootstrap sample (random sample with replacement) of the dataset. On average, about 63% of the training data is included in each bootstrap sample, leaving approximately 37% as **Out-of-Bag (OOB)** data.

Procedure

1. For each tree, record the data points not included in the bootstrap sample (OOB samples).
2. After training, use each tree to predict its corresponding OOB samples.
3. For each training instance, collect the predictions from all trees for which the instance was OOB.
4. Aggregate the OOB predictions (e.g., majority vote for classification, average for regression).
5. Compare aggregated OOB predictions to the true labels to compute OOB error.

Advantages

- Provides an unbiased estimate of model performance.
- Reduces the need for a separate validation set.
- Useful for model tuning and early stopping.

Limitations

- Less reliable for small datasets (insufficient OOB samples per instance).
- OOB error may not reflect performance on very different test distributions.

Formula (for classification)

$$\text{OOB Error Rate} = \frac{1}{N} \sum_{i=1}^{N} \nVdash(\hat{y}_i^{\text{OOB}} \neq y_i)$$

Where:

- N is the number of training samples
- $\hat{y}_i^{\text{OOB}}$ is the aggregated prediction for instance i from trees that did not train on it
- y_i is the true label
- $\nVdash$ is the indicator function

Advantages

- Reduces overfitting compared to single decision trees
- Handles both classification and regression
- Robust to noise and outliers
- Works well with high-dimensional data
- No need for feature scaling

Disadvantages

- Less interpretable than a single decision tree
- Large number of trees increases training and prediction time
- May not perform well on very sparse or low-signal datasets

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, non-parametric machine learning algorithm used for both **classification** and **regression**. It makes predictions based on the k closest training samples in the feature space. KNN does not learn an explicit model; instead, it relies entirely on the stored training data during prediction time.

Key Features

- **Lazy learner:** No model is trained; all computation happens during prediction.
- **Distance-based:** The algorithm uses a distance metric to find the most similar points.
- **Sensitive to feature scale:** All features should be scaled (e.g., using standardization or normalization) to ensure fair distance comparisons.

Distance Metrics

- **Euclidean Distance (L2 norm):**

  $$d(x, x') = \sqrt{\sum_{i=1}^{n} (x_i - x'_i)^2}$$

  Use when: Your features are continuous (e.g., height, weight, pixel values), and the data follows a geometric structure. This is the default and works well when features are on the same scale.

- **Manhattan Distance (L1 norm):**

  $$d(x, x') = \sum_{i=1}^{n} |x_i - x'_i|$$

  Use when: Features are sparse, ordinal, or represent grid-like movement (e.g., text frequency, city block distances). More robust to outliers than Euclidean.

- **Minkowski Distance (generalized form):**

  $$d(x, x') = \left(\sum_{i=1}^{n} |x_i - x'_i|^p \right)^{1/p}$$

  Use when: You want a flexible metric. Setting $p = 1$ gives Manhattan distance, $p = 2$ gives Euclidean. Larger p puts more emphasis on larger differences.

KNN for Classification

KNN is widely used for classification tasks where the output is a discrete label. It predicts the class of a new sample by majority vote among its k nearest neighbors.

Steps:

1. Calculate the distance between the query point x and all training samples.
2. Select the k training samples with the smallest distances.
3. Retrieve the class labels $y_1, y_2, \ldots, y_k$ of the k nearest neighbors.
4. Count the frequency of each class label among the k neighbors.
5. Assign the most frequent class as the predicted label.

Prediction:

Let $\mathcal{N}_k(x)$ be the set of k nearest neighbors of the input point x, and y_i the label of the i-th neighbor. The prediction is:

$$\hat{y} = \arg\max_{c \in \mathcal{C}} \sum_{i \in \mathcal{N}_k(x)} \mathbb{I}(y_i = c)$$

Where:

- $\mathcal{C}$ is the set of all possible classes,
- $\mathbb{I}(\cdot)$ is the indicator function that returns 1 if the condition is true, else 0,
- The class with the highest count is chosen.

KNN for Regression

In regression, the goal is to predict a continuous value. KNN regression works by averaging the target values of the k nearest neighbors.

Steps:

1. Calculate the distance between the query point x and all training samples.
2. Select the k training samples with the smallest distances.
3. Retrieve the target values $y_1, y_2, \ldots, y_k$ of these k neighbors.
4. Compute the average of these target values.

Prediction:

Let $\mathcal{N}_k(x)$ be the indices of the k nearest neighbors of input x. The predicted value is the mean:

$$\hat{y} = \frac{1}{k} \sum_{i \in \mathcal{N}_k(x)} y_i$$

Where:

- y_i is the target value of the i-th nearest neighbor.
- This simple average can also be replaced with a weighted average (e.g., inverse distance weighting) for better accuracy in some scenarios.

Hyperparameters

- k: Number of neighbors (odd for binary classification to avoid ties)
- Distance metric (Euclidean, Manhattan, etc.)
- `weights`: 'uniform' (equal weight) or 'distance' (closer neighbors have higher influence)
- `algorithm`: 'brute', 'kd_tree', or 'ball_tree' for nearest neighbor search

Feature Scaling

KNN is sensitive to the scale of features. It's recommended to standardize or normalize features:

$$x_i' = \frac{x_i - \mu_i}{\sigma_i} \quad \text{or} \quad x_i' = \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$$

Advantages

- Simple to implement and understand
- No assumptions about data distribution
- Effective when decision boundary is irregular

Disadvantages

- Slow prediction for large datasets (computationally expensive)
- Memory-intensive (stores all training data)
- Performance degrades with high dimensionality (curse of dimensionality)
- Sensitive to irrelevant or redundant features

K-Means Clustering

K-Means is an unsupervised clustering algorithm that partitions data into K distinct non-overlapping clusters. Each cluster is defined by its centroid (mean of points in that cluster).

Objective

Minimize the total within-cluster sum of squared distances (WCSS) between points and their assigned cluster centroids:

$$J = \sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2$$

Where:

- K = number of clusters
- C_i = set of points in cluster i
- μ_i = centroid of cluster i

Algorithm Procedure

1. **Initialization:** Choose K initial centroids randomly (or use K-Means++ for better initialization).
2. **Assignment Step:** Assign each point to the nearest centroid:

$$C_i = \{x_j : \|x_j - \mu_i\|^2 \leq \|x_j - \mu_k\|^2, \forall k\}$$

3. **Update Step:** Recalculate the centroid of each cluster:

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

4. **Repeat:** Steps 2 and 3 until convergence (i.e., no change in assignments or centroids).

K-Means++ Initialization

K-Means++ is an improved method for initializing centroids that leads to better clustering results and faster convergence compared to random initialization. It helps mitigate the sensitivity of K-Means to initial centroid positions.

Procedure

The K-Means++ algorithm selects initial centroids as follows:

1. Choose the first centroid μ_1 randomly from the data points.
2. For each remaining data point x, compute its distance $D(x)$ to the nearest already chosen centroid.
3. Choose the next centroid from the remaining data points with probability proportional to $D(x)^2$. That is:
$$P(x) = \frac{D(x)^2}{\sum_{x'} D(x')^2}$$
4. Repeat steps 2–3 until K centroids have been chosen.

Advantages of K-Means++

- Reduces the chances of poor initialization leading to suboptimal clusters.
- Leads to faster convergence.
- Improves the stability and quality of the final clusters.

Convergence

K-Means is guaranteed to converge in a finite number of iterations. The objective function J decreases with each iteration, but convergence may be to a local minimum.

Hyperparameters

- K: Number of clusters (chosen using methods like the Elbow Method or Silhouette Score)
- Initialization method: 'random' or 'k-means++'
- Max iterations: Upper limit for convergence
- Tolerance: Minimum change in centroid movement for stopping

Feature Scaling

K-Means is distance-based, so features should be standardized:

$$x_i' = \frac{x_i - \mu_i}{\sigma_i}$$

Advantages

- Simple and easy to implement
- Scales well to large datasets
- Works well when clusters are spherical and evenly sized

Disadvantages

- Requires pre-specifying the number of clusters K
- Sensitive to initialization (can converge to local minima)
- Not suitable for non-spherical or unevenly sized clusters
- Sensitive to outliers

Common Evaluation Methods

- **Elbow Method:** Plot WCSS vs. K and find the 'elbow' point
- **Silhouette Score:** Measures how similar a point is to its own cluster vs. others
- **Davies-Bouldin Index**, **Calinski-Harabasz Index**

Naive Bayes

Naive Bayes is a probabilistic classifier based on Bayes' Theorem with a strong assumption of feature independence. It is commonly used for text classification, spam detection, and sentiment analysis.

Bayes' Theorem

$$P(y \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid y) \cdot P(y)}{P(\mathbf{x})}$$

Where:

- $P(y \mid \mathbf{x})$ = posterior probability of class y given features $\mathbf{x}$
- $P(\mathbf{x} \mid y)$ = likelihood of features given class y
- $P(y)$ = prior probability of class y
- $P(\mathbf{x})$ = evidence (constant across all classes)

Naive Independence Assumption

Assumes features are conditionally independent given the class:

$$P(\mathbf{x} \mid y) = \prod_{i=1}^{n} P(x_i \mid y)$$

So, the posterior becomes:

$$P(y \mid \mathbf{x}) \propto P(y) \cdot \prod_{i=1}^{n} P(x_i \mid y)$$

Classification Procedure

1. Estimate prior: $P(y) = \frac{\text{\# of examples in class } y}{\text{total \# of examples}}$
2. Estimate likelihood: $P(x_i \mid y)$ using appropriate distribution:
   - Categorical $\rightarrow$ frequency
   - Text $\rightarrow$ word count with Laplace smoothing
   - Continuous $\rightarrow$ Gaussian distribution
3. Compute the posterior for each class:

$$\hat{y} = \arg\max_{y} \left[P(y) \cdot \prod_{i=1}^{n} P(x_i \mid y) \right]$$

Types of Naive Bayes

- **Gaussian Naive Bayes:** Assumes that continuous features follow a Gaussian (normal) distribution. The likelihood of the feature x_i given class y is:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_{y,i}^2}} \exp\left(-\frac{(x_i - \mu_{y,i})^2}{2\sigma_{y,i}^2}\right)$$

  where $\mu_{y,i}$ and $\sigma_{y,i}^2$ are the mean and variance of feature x_i for class y.

- **Multinomial Naive Bayes:** Suitable for features representing counts, such as term frequencies in text classification. The likelihood is modeled using a multinomial distribution:

$$P(x \mid y) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_{i=1}^{n} \theta_{y,i}^{x_i}$$

  where $\theta_{y,i}$ is the probability of feature i occurring in class y, and x_i is the count of feature i.

- **Bernoulli Naive Bayes:** Used when features are binary (e.g., 1 if a word appears in the document, 0 otherwise). The likelihood is:

$$P(x \mid y) = \prod_{i=1}^{n} \theta_{y,i}^{x_i}(1 - \theta_{y,i})^{1-x_i}$$

  where $\theta_{y,i}$ is the probability that feature i is present in class y.

Assumptions

- Features are conditionally independent given the class label.
- Features are equally important.
- Each class has its own distribution for features.

Dealing with Violations:

- Independence rarely holds perfectly, but Naive Bayes still performs well in practice.
- Use more complex models (e.g., Tree-Augmented Naive Bayes) if performance is inadequate.

Advantages

- Simple, fast, and efficient even with large feature spaces
- Works well with high-dimensional data (e.g., text classification)
- Requires less training data

Disadvantages

- Strong independence assumption often unrealistic
- Can struggle with correlated features
- Probabilistic outputs may be unreliable in practice

Support Vector Machine (SVM)

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for both classification and regression tasks. It aims to find the optimal hyperplane that best separates the classes by maximizing the margin between them.

1. Key Idea

Find a hyperplane that maximally separates data points of different classes. The closest data points to the hyperplane are called **support vectors**, and the margin is defined as the distance between these support vectors and the hyperplane.

2. Linearly Separable Case (Hard Margin)

Objective Function

Given a dataset $\{(x_i, y_i)\}_{i=1}^n$, where $y_i \in \{-1, +1\}$:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}^\top x_i + b) \geq 1$$

Decision function:

$$f(x) = \text{sign}(\mathbf{w}^\top x + b)$$

3. Linearly Non-Separable Case (Soft Margin)

Use slack variables ξ_i to allow some misclassification:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^n \xi_i \quad \text{subject to} \quad y_i(\mathbf{w}^\top x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Where:

- C is a regularization parameter balancing margin width and misclassification penalty.

4. Kernel Trick

The kernel trick allows SVM to perform classification in higher-dimensional spaces without explicitly mapping the data. Instead of computing $\phi(x_i)^\top \phi(x_j)$, we compute $K(x_i, x_j)$ directly.

Common Kernels

- **Linear:** $K(x, x') = x^\top x'$ Best for linearly separable data or when the number of features is large.
- **Polynomial:** $K(x, x') = (x^\top x' + c)^d$ Useful when data shows polynomial relationships.
- **RBF (Gaussian):** $K(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$ Suitable for most non-linear problems and widely used as default.
- **Sigmoid:** $K(x, x') = \tanh(\alpha x^\top x' + c)$ Similar to neural networks, not commonly used due to instability.

When to Use Which Kernel

- Use **linear kernel** if the data is linearly separable or when interpretability is important.
- Use **RBF** when data has complex boundaries and you don't know the form of the separation.
- Use **polynomial kernel** when you suspect feature interactions.
- Avoid sigmoid unless experimenting with neural network-like behaviors.

5. Classification Procedure

1. Normalize/scale features.
2. Choose kernel and set hyperparameters (C, kernel-specific).
3. Solve optimization (typically dual form with QP or SMO).
4. Identify support vectors.
5. Predict using:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b\right)$$

6. Regression with SVM (SVR)

Goal: Fit a function within a margin of tolerance ϵ from actual targets while penalizing deviations outside this margin.

SVR Objective (Epsilon-Insensitive)

$$\min_{\mathbf{w},b,\xi_i,\xi_i^*} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

$$\text{subject to: } \begin{cases} y_i - (\mathbf{w}^\top x_i + b) \le \epsilon + \xi_i \\ (\mathbf{w}^\top x_i + b) - y_i \le \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases}$$

Decision Function:

$$f(x) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(x_i, x) + b$$

Key Parameters

- ϵ: Width of the margin where no penalty is given.
- C: Controls trade-off between margin size and tolerance to deviations.

7. Hyperparameters

- C: Regularization strength
- Kernel type and its parameters (e.g., γ for RBF, degree for polynomial)
- ϵ: In SVR, margin of tolerance

8. Advantages

- Effective in high-dimensional spaces
- Robust to overfitting with proper regularization
- Kernel trick allows for non-linear separation

9. Disadvantages

- Slow training for large datasets
- Requires feature scaling
- Sensitive to choice of hyperparameters
- No direct probabilistic interpretation (can be added post-hoc)

Classification Metrics

Confusion Matrix

A confusion matrix for binary classification is defined as:

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

1. Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{19}$$

Explanation: Measures overall correctness of the model's predictions.
When to use: Best used when classes are balanced and all errors have equal cost.

2. Precision

$$\text{Precision} = \frac{TP}{TP + FP} \tag{20}$$

Explanation: Measures the proportion of positive predictions that are actually correct.
When to use: Important when false positives are costly (e.g., spam detection).

3. Recall (Sensitivity)

$$\text{Recall} = \frac{TP}{TP + FN} \tag{21}$$

Explanation: Measures the ability of the model to find all relevant positive cases.
When to use: Important when false negatives are costly (e.g., medical diagnosis).

4. F1-Score

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{22}$$

Explanation: Harmonic mean of precision and recall.
When to use: Best when there is an uneven class distribution and a balance between precision and recall is needed.

5. Specificity

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{23}$$

Explanation: Measures the ability to correctly identify negative results.
When to use: Useful when correctly identifying the negative class is important.

6. ROC Curve and AUC

ROC Curve: Plot of True Positive Rate (TPR) vs. False Positive Rate (FPR).

$$\text{TPR} = \frac{TP}{TP + FN}, \quad \text{FPR} = \frac{FP}{FP + TN} \tag{24}$$

AUC (Area Under Curve): Measures the model's ability to distinguish between classes.
When to use: Best used for comparing classifiers and when working with imbalanced data.

7. Log Loss (Cross-Entropy)

$$\text{Log Loss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right] \tag{25}$$

Explanation: Evaluates the quality of the predicted probabilities.
When to use: Useful when confidence in classification matters (e.g., probabilistic models).

8. Matthews Correlation Coefficient (MCC)

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{26}$$

Explanation: Provides a balanced measure even if the classes are of very different sizes.
When to use: Suitable for imbalanced datasets and when you want a single, informative metric.

9. Multi-class Evaluation

Macro Average: Averages metric independently for each class — treats all classes equally.
Weighted Average: Averages metric by taking class imbalance into account — more robust for imbalanced classes.

10. Hamming Loss (for Multi-label)

$$\text{Hamming Loss} = \frac{1}{n \cdot L}\sum_{i=1}^{n}\sum_{j=1}^{L}\mathbf{1}(y_{ij} \neq \hat{y}_{ij}) \tag{27}$$

Explanation: Measures the fraction of incorrect labels over the total number of labels.
When to use: Applicable to multi-label classification problems where each instance can belong to multiple classes.

Regression Metrics

1. Mean Absolute Error (MAE)

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{28}$$

Explanation: Average of the absolute differences between actual and predicted values.
When to use: When all errors are equally important and less sensitivity to outliers is desired.

2. Mean Squared Error (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{29}$$

Explanation: Average of the squared differences between actual and predicted values.
When to use: When larger errors should be penalized more heavily.

3. Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{30}$$

Explanation: Square root of MSE; interpretable in the same units as the target variable.
When to use: Useful when you want to penalize large errors and retain interpretability in original scale.

4. R-Squared (R^2 Score)

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{31}$$

Explanation: Proportion of variance in the target variable that is predictable from the features.
When to use: To assess overall model performance relative to a naive baseline (mean of target values).

5. Adjusted R-Squared

$$\text{Adjusted } R^2 = 1 - \left(\frac{(1 - R^2)(n - 1)}{n - k - 1}\right) \tag{32}$$

Explanation: Modifies R^2 to account for the number of predictors in the model.
When to use: Use when comparing models with different numbers of predictors.

6. Mean Absolute Percentage Error (MAPE)

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{33}$$

Explanation: Average of the absolute percentage errors.
When to use: Useful for interpretability in percentage terms, but avoid when $y_i = 0$ or near-zero.

7. Huber Loss

$$L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{for } |y - \hat{y}| \le \delta \\ \delta \cdot (|y - \hat{y}| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

Explanation: Combines MAE and MSE — behaves like MSE for small errors and MAE for large errors.
When to use: When you want to reduce the influence of outliers while still penalizing large deviations.

Clustering Metrics

1. Silhouette Score

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{34}$$

Where:

- $a(i)$ = average distance between point i and other points in the same cluster.
- $b(i)$ = minimum average distance between point i and all points in other clusters.

Explanation: Measures how similar a point is to its own cluster compared to other clusters. Values range from -1 to 1.
When to use: Use when you want to evaluate clustering quality without ground truth labels.

2. Davies-Bouldin Index

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)} \right) \tag{35}$$

Where:

- σ_i = average distance of points in cluster i to centroid c_i.
- $d(c_i, c_j)$ = distance between centroids of clusters i and j.

Explanation: Lower values indicate better clustering — compact and well-separated clusters.
When to use: Use when you need to compare multiple clustering models without labels.

3. Calinski-Harabasz Index (Variance Ratio Criterion)

$$CH = \frac{\text{Tr}(B_k)}{\text{Tr}(W_k)} \cdot \frac{n-k}{k-1} \tag{36}$$

Where:

- B_k = between-cluster dispersion matrix.
- W_k = within-cluster dispersion matrix.
- n = number of data points.
- k = number of clusters.

Explanation: Higher scores imply dense and well-separated clusters.
When to use: Good for comparing models where number of clusters vary.

4. Inertia (Within-Cluster Sum of Squares)

$$\text{Inertia} = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2 \tag{37}$$

Where:

- μ_i = centroid of cluster C_i.

Explanation: Measures cluster compactness. Lower inertia means tighter clusters.
When to use: Commonly used in K-Means. Useful in elbow method to choose optimal number of clusters.

5. Adjusted Rand Index (ARI)

$$ARI = \frac{\text{Index} - \text{Expected Index}}{\text{Max Index} - \text{Expected Index}} \tag{38}$$

Explanation: Compares clustering result to ground truth. Corrects for chance. Ranges from -1 to 1.
When to use: Use when true labels are available for validation.

6. Normalized Mutual Information (NMI)

$$NMI(U, V) = \frac{2 \cdot I(U; V)}{H(U) + H(V)} \tag{39}$$

Where:

- $I(U; V)$ = mutual information between cluster assignments U and true labels V.

- $H(U)$ and $H(V)$ = entropy of the clusters and labels.

Explanation: Measures shared information between predicted and true labels. Ranges from 0 to 1.
When to use: Use when you have ground truth labels and want to evaluate consistency with clustering output.

Feature Scaling Techniques

1. Standardization (Z-score Normalization)

Formula:

$$z = \frac{x - \mu}{\sigma} \tag{40}$$

Explanation: Transforms the data to have zero mean and unit variance.
When to use: Use when features are normally distributed. Commonly used with algorithms like Logistic Regression, SVM, KNN, and PCA.

2. Min-Max Scaling (Normalization)

Formula:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{41}$$

Explanation: Rescales the feature to a fixed range, usually $[0, 1]$.
When to use: Use when you want to preserve the shape of the original distribution and scale features to the same range (e.g., for Neural Networks).

3. Robust Scaling

Formula:

$$x' = \frac{x - \text{median}(x)}{\text{IQR}(x)} \tag{42}$$

Explanation: Uses the median and interquartile range (IQR) to scale the data, making it robust to outliers.
When to use: Use when the data contains many outliers or is skewed.

Distance Metrics

1. Euclidean Distance

Formula:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{43}$$

Explanation: Measures the straight-line (L2 norm) distance between two points in Euclidean space.
When to use: Use when features are continuous and scaled. Common in KNN, K-Means, clustering.

2. Manhattan Distance

Formula:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{44}$$

Explanation: Also called L1 norm or ”taxicab” distance; measures the path along grid lines.
When to use: Use when dealing with high-dimensional or sparse data; less sensitive to outliers than Euclidean.

3. Minkowski Distance

Formula:

$$d(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{45}$$

Explanation: Generalized distance metric. When $p = 1$, it's Manhattan; when $p = 2$, it's Euclidean.
When to use: Use when you want flexibility to adjust sensitivity with parameter p.

4. Cosine Similarity (Distance)

Formula:

$$\text{Similarity}(x,y) = \frac{x \cdot y}{\|x\| \|y\|}, \quad \text{Distance} = 1 - \text{Similarity} \tag{46}$$

Explanation: Measures the angle between two vectors (ignores magnitude).
When to use: Use when magnitude doesn't matter (e.g., text mining, TF-IDF vectors).

5. Hamming Distance

Formula:

$$d(x,y) = \sum_{i=1}^{n} \mathbb{I}(x_i \neq y_i) \tag{47}$$

Explanation: Counts the number of positions at which two binary strings differ.
When to use: Use with categorical or binary features.