Machine Learning with Python

Contents

1 Overview

Machine learning is a subset of AI that uses computer algorithms that require feature engineering. It teaches computers to learn from data and identify patterns and use them to make decisions without receiving explicit input from the user. There are three main types of machine learning models:

1. Supervised learning models are trained on a known set of features and a target variable, to identify relationships which are then used for inference or forecasting on new data. For example: linear regression.
2. Unsupervised learning models do not admit labelled feature-target style data, instead they are trained on a set of variables which are all considered features; the model finds relationships between these features. For example: Principal Component Analysis.
3. Reinforcement learning models simulate an AI agent interacting with its environment, they learn how to make decisions based on feedback from the environment. For example:

The two focuses of supervised learning are regression and classification One of the main types of unsupervised learning are clustering

1.1 Machine Learning Lifecycle

1. **Problem Definition:** Clearly state the objective and desired outcome.
2. **Data Collection:** Identify required data and its source.
3. **Data Preparation:** Clean the data, handle missing values, normalize if necessary, engineer features, and perform exploratory data analysis. Split into training and testing sets.
4. **Model Development:** Train the model, tune hyperparameters, and evaluate performance using appropriate metrics.
5. **Deployment:** Integrate the trained model into a production environment.

1.2 Difference between a Data Scientist and AI Engineer

Aspect	Data Science	AI Engineering
Primary Use Cases	Descriptive and predictive analytics (e.g., EDA, clustering, regression, classification)	Prescriptive and generative AI (e.g., optimisation, recommendation systems, intelligent assistants)
Data Type Focus	Primarily structured/tabular data, cleaned and preprocessed	Primarily unstructured data (text, images, audio, video), used at large scale
Model Characteristics	Narrow-scope ML models, smaller in size, domain-specific, faster to train	Foundation models, large-scale, general-purpose, high compute and data requirements
Development Process	Data-driven model development (feature engineering, training, validation)	Application of pre-trained models with prompt engineering, PEFT, and RAG frameworks

Table 1: Key Differences Between Data Science and AI Engineering

1.3 Tools for Machine Learning

Python has several modules that handle the different stages of the machine learning model development pipeline:

1. Data preprocessing: `pysql`, `pandas`
2. Exploratory data analysis: `pandas`, `numpy`, `matplotlib`
3. Optimisation: `scipy`
4. Implementation: `scikit-learn` (supervised and unsupervised methods), `keras`, `pytorch` (deep learing)

1.4 Scikit-learn

The basic syntax for using supervised learning models in `scikit-learn` follows a standard workflow:

1. **Split the data:**

   ```
   x_train, X_test, y_train, y_test = train_test_split(X, y, test_size=...)
   ```

2. **Import and initialise the model:**

   ```
   from sklearn import svm
   model = svm.SVC(...)
   ```

3. **Train the model:**

   ```
   model.fit(X_train, y_train)
   ```

4. **Make predictions:**

   ```
   predictions = model.predict(X_test)
   ```

5. **Evaluate performance:** Use a confusion matrix to assess classification accuracy:

   ```
   from sklearn.metrics import confusion_matrix
   confusion_matrix(y_test, predictions)
   ```

6. **Optional – Save the model:** You can serialise the trained model using `pickle`:

   ```
   import pickle
   with open("model.pkl", "wb") as f:
       pickle.dump(model, f)
   ```

   Whether this step is necessary depends on the context and industry practice.

2 Linear Regression

Regression is a type of supervised learning model. It models a relationship between a continuous target variable and explanatory features.

2.1 Linear Regression

Definition

Linear regression models the relationship between a response variable Y and one or more features $X_1, \ldots, X_p$. In the case of simple linear regression with one predictor X, the model assumes:

$$Y \approx \beta_0 + \beta_1 X$$

where β_0 is the intercept and β_1 is the slope.

Coefficient Estimation

Given training data $(x_1, y_1), \ldots, (x_n, y_n)$, the goal is to estimate the coefficients β_0, β_1 such that the fitted values are as close as possible to the observed values. This is done by minimizing a norm of the residual vector:

$$\beta_{\min} = \arg\min_{\beta \in \mathbb{R}^2} \|\mathbf{Y} - \mathbf{X}\beta\|_p^p$$

For ordinary least squares (OLS), we use the 2-norm ($p = 2$), leading to the minimisation of the residual sum of squares (RSS). The OLS solution yields closed-form expressions for $\hat{\beta}_0$ and $\hat{\beta}_1$. For least absolute deviations (LAD), we take $p = 1$. For Chebyshev or minimax regression, we take $p = \infty$, minimising the maximum residual.

Population vs Sample Regression

In the real world we almost always do not have a maximal data set. This means that we do not have data for every single item in the population, all we can hope to obtain are samples. The fitted regression line from a sample *estimates* the *population regression line*:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

where ε is the irreducible error due to unobserved factors. Since we cannot observe the full population, we fit the model on a sample and obtain estimates $\hat{\beta}_0, \hat{\beta}_1$. However, since we only observe a sample, the estimates $\hat{\beta}$ vary from sample to sample[1]. The sample regression is thus an estimate of the population regression. If we had access to all population data, the OLS minimisation would yield the true β_0, β_1.

Sampling and the Central Limit Theorem

Across repeated samples of size n, we obtain different estimates of β, say $\hat{\beta}^{(1)}, \ldots, \hat{\beta}^{(n)}$. Their average converges to the true coefficient $\bar{\beta}$ as $n \to \infty$, due to the central limit theorem:

$$\hat{\beta} - \bar{\beta} \xrightarrow{d} \mathcal{N}(0, \sigma_*)$$

Standard Error and Inference

The variance of the coefficient estimates across samples is the *standard error*:

$$SE(\hat{\beta}_i) = f(\sigma^2, x_j, \bar{x})$$

where σ^2 is estimated from the data via:

$$\hat{\sigma}^2 = \frac{RSS}{n-2}$$

The standard error quantifies the variability of the estimated coefficient under repeated sampling. If we observe a high standard error then it means that we do not see replicability of the relationship as we vary our sample, which may suggest that the relationship between the features and target is ephemeral and noisy, so any forecast or inference made using it will be unreliable.

2.2 Multiple Linear Regression

Model Definition

Multiple linear regression generalises simple linear regression by modelling a response variable Y as a linear combination of multiple predictors $X_1, \ldots, X_p$:

$$\hat{Y} = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \cdots + \theta_p X_p = \mathbf{X}\boldsymbol{\theta}$$

where:

- $\mathbf{X}$ is the design matrix including a column of ones for the intercept,
- $\boldsymbol{\theta}$ is the parameter vector including θ_0 (intercept) and $\theta_1, \ldots, \theta_p$ (slopes).

For $p = 1$, the model defines a line as with the usual linear regression, for $p = 2$, the model defines a plane, for $p > 2$, the model defines a hyperplane in $\mathbb{R}^{p+1}$.

Estimation via Least Squares

As before the $\boldsymbol{\theta}$ coefficients are the population coefficients, but in reality we can only estimate these. These are the sample coefficients $\hat{\boldsymbol{\theta}}$ which are obtained by minimising the mean squared error (MSE), but there is no closed form solution to this minimisation problem and so numerical methods are used. For large datasets, iterative methods such as gradient descent can also be used to minimise MSE.

Prediction and Residuals

Predictions for a new data point $\mathbf{x}$ are given by:

$$\hat{y} = \mathbf{x}^\top \hat{\boldsymbol{\theta}}$$

The residual error for observation i is:

$$\varepsilon_i = y_i - \hat{y}_i$$

The squared residuals are used in minimising the MSE and finding the optimim sample coefficients.

[1] The goal of statistical inference is to understand how close $\hat{\beta}$ is to the true β and quantify this uncertainty.

Model Complexity and Overfitting

Including more variables increases model flexibility but can lead to overfitting, where the model captures noise rather than signal. A balance is needed:

- Remove redundant, highly correlated (collinear) variables.
- Choose variables that are interpretable, uncorrelated, and strongly related to Y.

Categorical Variables

Categorical predictors must be encoded numerically:

- Binary variables: encoded as 0/1.
- Multiclass variables: use one-hot encoding (create a dummy variable for each class).

What-If Analysis

The model can be used for counterfactual predictions by altering input features. However, what-if analysis may be invalid if:

- Scenarios lie far outside the training data (extrapolation),
- Variables are collinear — changing one realistically requires changing another.

2.3 Logistic Regression

Model Definition

Logistic regression models the probability that a binary outcome $Y \in \{0, 1\}$ occurs, given features $X_1, \ldots, X_p$. The model assumes that the log-odds (logit) of the probability is a linear function of the input:

$$\log\left(\frac{\mathbb{P}(Y = 1 \mid X)}{\mathbb{P}(Y = 0 \mid X)}\right) = \theta_0 + \theta_1 X_1 + \cdots + \theta_p X_p = \mathbf{X}^\top \boldsymbol{\theta}$$

Solving for the probability gives the sigmoid function:

$$\hat{p} = \mathbb{P}(Y = 1 \mid X) = \sigma(\mathbf{X}^\top \boldsymbol{\theta}) = \frac{1}{1 + e^{-\mathbf{X}^\top \boldsymbol{\theta}}}$$

This maps the linear combination of features to the interval $(0, 1)$, giving a probability.

Learning the Parameters

To fit the model onto data we use maximum likelihood estimation to learn the $\boldsymbol{\theta}$ parameters. The optimal parameters $\hat{\boldsymbol{\theta}}$ are found by solving:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$$

where $\mathcal{J}$ is negative log likelihood. This optimisation is typically performed using gradient descent or related numerical techniques.

Prediction and Decision Boundary

The model returns a probability $\hat{p}$. To make a classification, we choose a threshold $\tau \in (0, 1)$ (which is usually $\tau = 0.5$) and define:

$$\hat{Y} = \begin{cases} 1 & \text{if } \hat{p} \geq \tau \\ 0 & \text{otherwise} \end{cases}$$

This tells us that if $\mathbb{P}(Y = 1|X) < 0.5$ then $\hat{Y}$ is classified as belonging to class 0. The threshold can be fine tuned to match reality, since most phenomena are not cleanly 50:50. For example fraud may occur one time out of 99 so using a 0.5 threshold would clearly be unwise since it would class everything as not-fraud.

Interpretability

Each coefficient θ_j measures the change in the log-odds of $Y = 1$ for a unit increase in X_j, holding other variables fixed. Larger magnitudes indicate stronger influence. Importantly, this allows us to conduct inference involving continuous change on a discrete classification problem.

Use Cases

Logistic regression is used when:

- The target variable is binary.
- Probabilities, not just classifications, are required.
- Model interpretability is important.

2.4 Nonlinear and Polynomial Regression

Motivation

In many real-world datasets, the relationship between features X and the target Y is not well captured by a straight line. Nonlinear regression models are used when such relationships require curvature or more complex forms.

Nonlinear Regression: Definition

Nonlinear regression models the relationship between Y and the inputs $X_1, \ldots, X_p$ via a nonlinear function:

$$\hat{Y} = f(X_1, \ldots, X_p; \boldsymbol{\theta})$$

where f is nonlinear in its parameters $\boldsymbol{\theta}$. Common forms include:

- Exponential: $\hat{Y} = \theta_0 e^{\theta_1 X}$
- Logarithmic: $\hat{Y} = \theta_0 + \theta_1 \log(X)$
- Sinusoidal: $\hat{Y} = \theta_0 + \theta_1 \sin(\theta_2 X)$

Polynomial Regression

Polynomial regression models Y as an n-th degree polynomial in a single variable X:

$$\hat{Y} = \theta_0 + \theta_1 X + \theta_2 X^2 + \cdots + \theta_n X^n$$

This model is nonlinear in the input but linear in the parameters, so it can be recast as:

$$\hat{Y} = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \cdots + \theta_n X_n$$

where $X_k = X^k$, allowing the use of ordinary least squares.

Fitting and Optimisation

For models linear in parameters (e.g., polynomial), parameters $\boldsymbol{\theta}$ can be fitted using standard linear regression:

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

For truly nonlinear models, where f is nonlinear in $\boldsymbol{\theta}$, closed-form solutions do not exist. Instead, we minimise a loss (usually MSE) using iterative optimisation techniques, such as:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} (y_i - f(x_i; \boldsymbol{\theta}))^2$$

This is solved numerically using algorithms like gradient descent.

Model Selection and Overfitting

Polynomial models of high degree can always perfectly fit any finite dataset (interpolate all points), but this leads to overfitting. Overfit models capture noise rather than signal.

Visual Model Selection

To identify whether a nonlinear model is needed:

- Plot scatter diagrams of Y vs. X_i
- Look for nonlinearity (e.g., curvature, saturation, periodicity)
- Choose candidate functions: polynomial, exponential, logarithmic, etc.

Examples of Nonlinear Forms

- Exponential growth: GDP, investment returns
- Logarithmic growth: Diminishing returns in productivity
- Sinusoidal: Seasonal patterns, temperature cycles

Beyond Parametric Forms

When no closed-form function is known, or when relationships are nonlinear and defy any known non-linear archetypes, meaning we cannot hope to represent the relationship via some standard paremetric, mathematical equation, we require more advanced methods such as

- Decision trees
- Random forests
- Support vector machines
- Neural networks
- k-nearest neighbours
- Gradient boosting

These are non-parametric regressors that adapt to complex patterns without requiring an explicit functional form.

3 Tree-based Methods

3.1 Decision Trees

A **decision tree** is a supervised learning algorithm used for classification. It models decisions as a tree-like structure, where:

- Internal nodes represent tests on input features,
- Branches represent outcomes of those tests,
- Leaf nodes assign a class (for classification) or a value (for regression).

The flow from root to leaf encodes a sequence of decisions that leads to a prediction.

Training: Recursive Partitioning

A decision tree is trained by recursively partitioning the feature space to reduce uncertainty in the target variable. The training process proceeds as follows:

1. Start with all training data at the root.
2. At each node:
    - Evaluate all possible feature-based splits,
    - Select the split that maximally reduces impurity,
    - Partition the data accordingly and assign subsets to new child nodes.
3. Repeat recursively for each child node.
4. Stop splitting when:
    - All samples in a node belong to the same class,
    - No features remain,
    - A stopping condition is met.

This process is known as **recursive binary splitting**.

Split Selection Criteria

At each node, the goal is to select the feature and threshold that most reduces the impurity of the child nodes.

1. **Entropy (Information Gain)**: Entropy measures the randomness of the class distribution:

$$\text{Entropy}(t) = -\sum_k p_k \log_2(p_k)$$

where p_k is the proportion of class k in node t. The split that maximises the **Information Gain** is chosen:

$$\text{IG} = \text{Entropy(parent)} - \sum_i \frac{n_i}{n} \cdot \text{Entropy(child}_i\text{)}$$

2. **Gini Impurity**: An alternative impurity metric defined as:

$$\text{Gini}(t) = 1 - \sum_k p_k^2$$

Lower Gini implies purer nodes i.e more optimally split nodes. The best split minimises weighted Gini impurity.

Stopping and Pruning

To avoid overfitting, tree growth must be controlled.

1. **Pre-pruning (early stopping)**: Stop growing the tree when:
   - Maximum tree depth is reached,
   - A minimum number of samples per node or per leaf is exceeded,
   - A maximum number of leaves is reached.
2. **Post-pruning (reduction)**: Grow the full tree, then prune branches that do not improve performance on a validation set.

Pruning simplifies the model, improves generalisation, and increases interpretability.

Interpretability

Decision trees are highly interpretable models:

- They can be visualised as flowcharts,
- Feature importance is revealed by the order of splits,
- Each path from root to leaf corresponds to a decision rule.

3.2 Regression Trees

A **regression tree** is a variant of the decision tree used for predicting continuous target variables rather than categorical labels. While classification trees assign inputs to discrete classes, regression trees predict a numerical value, typically by averaging the target values in a leaf node.

Problem Setup

- **Classification tree:** Target variable Y is categorical; prediction is a class label.
- **Regression tree:** Target variable Y is continuous; prediction is a real number $\hat{y}$.

The structure of the tree is the same as in classification: internal nodes represent decisions on input features, branches correspond to outcomes of those decisions, and leaves output predictions.

Training: Recursive Splitting

The tree is built recursively by splitting the data to reduce variability in the target variable. At each node:

1. For each feature, evaluate candidate thresholds to split the data into left and right subsets.
2. For each candidate split, compute the **Mean Squared Error (MSE)**:

$$\text{MSE}(t) = \frac{1}{n_t} \sum_{i=1}^{n_t} (y_i - \bar{y}_t)^2$$

   where n_t is the number of observations in node t, and $\bar{y}_t$ is the mean of their target values.

3. Compute the **weighted average MSE** of the split:

$$\text{Split MSE} = \frac{n_L}{n} \text{MSE}_L + \frac{n_R}{n} \text{MSE}_R$$

   where n_L and n_R are the sample sizes in the left and right nodes respectively.

4. Select the feature and threshold with the lowest split MSE.
5. Recurse on child nodes until a stopping criterion is met.

Prediction Rule

At inference time, each data point traverses the tree until it reaches a leaf. The predicted value $\hat{y}$ is:

$$\hat{y} = \frac{1}{n_{\text{leaf}}} \sum_{i \in \text{leaf}} y_i$$

Other statistics (e.g., median) can be used if the target distribution is skewed, though the mean is most common due to computational efficiency.

Split Evaluation for Feature Types

- **Continuous features:** Trial thresholds α are defined as midpoints between sorted unique values. Exhaustive search yields the threshold that minimises weighted MSE.
- **Binary features:** Simply split on the two categories and compute the resulting split MSE.
- **Multi-class features:** Use strategies like one-vs-one or one-vs-all to reduce to binary splits, and evaluate each using the same MSE criterion.

Stopping Criteria and Pruning

As with classification trees, growth can be controlled using:

- Maximum depth,
- Minimum samples per node or per leaf,
- Maximum number of leaves.

Post-pruning may also be applied to remove splits that do not significantly reduce prediction error on a validation set.

Applications

Regression trees are widely used in domains requiring interpretable predictions of continuous outcomes, such as:

- Forecasting revenue,
- Predicting temperature or energy usage,
- Modelling medical risk scores or patient vitals,
- Estimating economic indicators.

4 Support Vector Machines (SVM)

Support Vector Machines (SVM) are supervised learning algorithms used for both classification and regression. The central idea is to represent data in a high-dimensional space and construct a separating hyperplane that best divides the data into classes (or fits a continuous target variable, in the case of regression).

4.1 Linear SVM for Binary Classification

Geometric Interpretation

In the binary classification setting, SVM aims to find the optimal hyperplane that separates the data into two classes with the maximum margin. Given two linearly separable classes, the separating hyperplane is defined as:

$$\mathbf{w}^\top \mathbf{x} + b = 0$$

where $\mathbf{w}$ is the normal vector to the hyperplane, and b is the bias. The margin is the distance from the hyperplane to the nearest data point of either class. The points that lie closest to the hyperplane are called **support vectors**.

Optimisation Objective

The optimisation problem is:

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

This formulation ensures that all points are correctly classified w ith a margin of at least 1, and $\|\mathbf{w}\|$ is minimised, thereby maximising the margin.

Soft Margin SVM

In real-world data, perfect separation is mostly not possible. **Soft margin SVM** introduces slack variables to allow misclassifications:

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

The parameter $C > 0$ controls the trade-off between maximising the margin and penalising misclassified points.

4.2 Kernels and Nonlinear Decision Boundaries

When data is not linearly separable in its original feature space, SVM employs the **kernel trick** to implicitly map data into a higher-dimensional space where a linear separation is possible. Common kernels include:

- **Linear kernel:** $K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$
- **Polynomial kernel:** $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}' + c)^d$
- **Radial Basis Function (RBF) kernel:** $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2\right)$
- **Sigmoid kernel:** $K(\mathbf{x}, \mathbf{x}') = \tanh(\alpha \mathbf{x}^\top \mathbf{x}' + c)$

These enable SVM to construct complex, nonlinear decision boundaries while retaining convex optimisation.

4.3 Support Vector Regression (SVR)

SVM can also be adapted for regression tasks through **Support Vector Regression (SVR)**.

Idea

Instead of trying to classify, SVR aims to fit a function that deviates from the observed targets by no more than a user-defined threshold ε. The region within which no penalty is applied is called the ε**-tube**.

Optimisation Formulation

The optimisation objective becomes:

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

subject to:

$$y_i - \mathbf{w}^\top \mathbf{x}_i - b \leq \varepsilon + \xi_i$$
$$\mathbf{w}^\top \mathbf{x}_i + b - y_i \leq \varepsilon + \xi_i^*$$
$$\xi_i, \xi_i^* \geq 0$$

Here, ε controls the tolerance zone, and C balances flatness vs tolerance of errors.

4.4 Advantages and Limitations

Advantages:

- Effective in high-dimensional spaces.
- Robust to overfitting (especially with margin control).
- Flexible via kernel functions for nonlinear problems.

Limitations:

- Training can be slow on large datasets.
- Sensitive to noisy data and overlapping classes.
- Performance depends heavily on kernel and hyperparameter selection.

4.5 Use Cases

SVMs are widely used in:

- Image classification (e.g., digit recognition)
- Spam filtering
- Text and sentiment analysis
- Speech recognition and anomaly detection
- Bioinformatics and pattern recognition

5 K-Nearest Neighbours (KNN)

K-Nearest Neighbours (KNN) is a supervised learning algorithm used for both classification and regression. It operates on the assumption that data points close to one another in the feature space are likely to share similar labels. Unlike other models, KNN is a **lazy learner**: it does not build an explicit model during training but instead stores the dataset and performs computation at prediction time.

5.1 Algorithmic Procedure

Given a value of k, and a new data point (the *query*), the KNN algorithm proceeds as follows:

1. Compute the distance from the query point to all labelled training data.
2. Select the k nearest data points.
3. **Classification:** predict the class that occurs most frequently among the k neighbours (majority vote).
4. **Regression:** predict the average (or median) of the target values of the k neighbours.

Common distance metrics include Euclidean, Manhattan, and Chebyshev distance (the p-norms with $p = 2, 1, \infty$). The choice of metric affects the neighbourhood geometry and, hence, prediction accuracy.

5.2 Decision Boundary

The **decision boundary** of a KNN classifier is shaped by the spatial distribution of training data. For low values of k, the boundary is highly irregular, following the noise in the data. As k increases, the boundary becomes smoother, leading to more generalised predictions.

5.3 Hyperparameter Tuning and Bias–Variance Trade-off

- **Small k:** Low bias, high variance. The model is highly sensitive to noise (overfitting).
- **Large k:** High bias, low variance. The model smooths over fine details (underfitting).

The optimal k is usually found via cross-validation.

5.4 Feature Scaling and Relevance

KNN is sensitive to the scale of features because distance metrics are directly affected by magnitude. Features should be standardised (e.g., via z-score normalisation) before applying KNN. Furthermore, including irrelevant or redundant features can degrade performance:

- Noisy features increase the required k to achieve stability.
- Irrelevant features dilute meaningful distances.
- Domain knowledge is crucial in selecting relevant features.

5.5 Weighted KNN

To address issues such as class imbalance or outliers, **distance-weighted KNN** assigns weights to neighbours inversely proportional to their distance from the query point. Closer points have greater influence:

$$w_i = \frac{1}{d(x, x_i)^2}$$

where x is the query point and x_i is a neighbour.

5.6 Regression with KNN

In KNN regression, predictions are made by averaging the target values of the k nearest neighbours:

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

This can be modified to a weighted average using distance-based weights to reduce the influence of distant neighbours.

5.7 Advantages and Limitations

Advantages:

- Simple, intuitive, and easy to implement.
- Naturally handles multi-class classification.
- No training phase; all computation is deferred until prediction.

Limitations:

- Computationally expensive at prediction time.
- Sensitive to feature scaling and irrelevant features.
- Poor performance on high-dimensional data due to the curse of dimensionality.

5.8 Use Cases

KNN is well-suited to:

- Pattern recognition (e.g., handwritten digit classification)
- Recommender systems
- Medical diagnosis (e.g., classifying patient symptoms)
- Anomaly detection

6 Bias–Variance Tradeoff and Ensemble Methods

6.1 Bias and Variance: An Intuitive Framework

The concepts of **bias** and **variance** describe two key sources of error in supervised learning models.

- **Bias** measures the error introduced by approximating a complex real-world function with a simpler model. High bias models tend to underfit the data.
- **Variance** measures the model's sensitivity to fluctuations in the training data. High variance models tend to overfit.

6.2 Prediction Bias and Variance

- **Prediction bias** quantifies the average deviation of model predictions from the true target values. A model with zero bias produces predictions that are, on average, correct.
- **Prediction variance** quantifies how much model predictions vary across different samples. High variance reflects a model that overfits on ephemera.

6.3 The Tradeoff and Model Complexity

As model complexity increases:

- Bias decreases, since the model can fit training data more closely.
- Variance increases, since the model becomes sensitive to fluctuations and noise in the data.

The **bias–variance tradeoff** captures the tension between underfitting and overfitting. The optimal model lies at the point where the sum of squared bias and variance is minimized, balancing generalization and flexibility.

6.4 Weak vs Strong Learners

- A **weak learner** is a model that performs slightly better than random guessing. It typically exhibits *high bias* and *low variance*.
- A **strong learner** performs significantly better but often exhibits *low bias* and *high variance*.

Decision trees are frequently used as weak learners in ensemble methods due to their tunable bias–variance properties.

6.5 Ensemble Methods

Ensemble learning combines multiple models (often weak learners) to produce a stronger predictor. Two primary techniques are **bagging** and **boosting**, which address bias and variance in different ways.

6.5.1 Bagging (Bootstrap Aggregating)

Bagging reduces variance without substantially increasing bias.

- It trains multiple base models (e.g., decision trees) on *bootstrap samples* of the data.
- The final prediction is the average (for regression) or majority vote (for classification) of the base models.

Bagging benefits from the law of large numbers: averaging multiple high-variance models leads to a lower-variance aggregate.

Random Forests Random Forests are a popular implementation of bagging:

- A large number of decision trees are trained on different bootstrap samples.
- During training, each node considers only a random subset of features.
- Final predictions are aggregated across trees.

This strategy further decorrelates the trees and reduces overfitting.

6.5.2 Boosting

Boosting reduces bias by sequentially training models that correct the errors of their predecessors.

- Each weak learner is trained on a modified dataset, with higher weights given to previously misclassified examples.
- The final model is a weighted sum of all learners.

Boosting increases model complexity iteratively, making it capable of approximating complex functions.

Popular Boosting Algorithms

- **AdaBoost:** Emphasizes misclassified examples by increasing their weights.
- **Gradient Boosting:** Models the residuals of prior models using gradient descent.
- **XGBoost:** An efficient and scalable implementation of gradient boosting.

6.6 Comparison of Bagging and Boosting

Aspect	Bagging	Boosting
Goal	Reduce variance	Reduce bias
Model training	Parallel	Sequential
Data sampling	Bootstrap (random)	Weighted (focused on errors)
Learner type	High variance, low bias	Low variance, high bias
Overfitting risk	Lower	Higher (if not regularized)
Example	Random Forests	AdaBoost, XGBoost

7 Unsupervised Learning

Unlike supervised learning, **unsupervised learning** deals with data that has no labels. The objective is to uncover hidden patterns, structures, or relationships within the data without any prior supervision. One of the most commonly used unsupervised learning algorithms is **K-Means Clustering**.

7.1 K-Means Clustering

K-Means is an iterative, centroid-based clustering algorithm that partitions a dataset into k non-overlapping clusters. It groups data points based on their similarity, measured via distance to cluster centroids.

7.1.1 Objective

The aim of K-Means is to minimise the *within-cluster sum of squares (WCSS)*, defined as:

$$\sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

where C_i is the i-th cluster and $\boldsymbol{\mu}_i$ is its centroid (mean).

7.1.2 Algorithm Steps

1. Choose the number of clusters k.
2. Initialise k centroids randomly.
3. Assign each data point to the nearest centroid using a distance metric (typically Euclidean).
4. Recalculate the centroids as the mean of all points assigned to each cluster.
5. Repeat steps 3 and 4 until convergence (i.e., centroids no longer move or a maximum number of iterations is reached).

7.1.3 Convergence and Performance

K-Means typically converges quickly and scales well to large datasets. However, performance is highly sensitive to:

- Initialisation of centroids
- The value of k
- Presence of noise or outliers
- Cluster shape and balance

7.1.4 Assumptions and Limitations

- Assumes clusters are convex and roughly equal in size.
- Sensitive to outliers and noise, as variance-based measures are easily distorted.
- Performs poorly when clusters differ significantly in size or density.
- Not suitable for discovering non-convex clusters.

7.1.5 Visual Example and Convergence

During convergence, centroids move to minimise intra-cluster distances. For example, in a dataset with two circular clusters, initial centroids placed randomly may gradually shift towards the true cluster centres over several iterations. Convergence is reached when the assignments no longer change.

7.1.6 Choosing the Number of Clusters

Selecting an appropriate k is non-trivial, especially in high-dimensional space. Common heuristic methods include:

- **Elbow Method:** Plot WCSS against k and look for the "elbow" point where additional clusters yield diminishing returns.
- **Silhouette Score:** Measures cohesion vs separation. Values close to 1 indicate well-separated clusters.
- **Davies-Bouldin Index:** Lower values indicate better clustering, based on intra- and inter-cluster distances.

7.1.7 Imbalanced and Overlapping Clusters

K-Means struggles with:

- **Imbalanced clusters:** Smaller clusters may be absorbed into larger ones.
- **Overlapping clusters:** When clusters are not well-separated, K-Means can mislabel data points.
- **Incorrect k:** If k is too small or too large, the resulting clustering may not reflect the true structure of the data.

7.2 DBSCAN and HDBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that identifies clusters as dense regions of points separated by areas of lower density. Unlike centroid-based methods (e.g., K-Means), DBSCAN does not assume convex cluster shapes and does not require the user to specify the number of clusters in advance.

7.2.1 Core Concepts

DBSCAN requires two key parameters:

- ε (epsilon): the radius defining a neighbourhood around a point.
- `minPts`: the minimum number of points required to form a dense region.

Based on these, each point is classified as one of the following:

- **Core Point:** A point with at least `minPts` points (including itself) within its ε-neighbourhood.
- **Border Point:** A point that lies within the ε-neighbourhood of a core point but is not itself a core point.
- **Noise Point:** A point that does not belong to the ε-neighbourhood of any core point.

7.2.2 How DBSCAN Works

1. Select an arbitrary unvisited point.
2. Retrieve its ε-neighbourhood.
3. If it contains at least `minPts`, start a new cluster.
4. Expand the cluster by recursively visiting all ε-neighbours of core points.
5. Label remaining unvisited points as noise.

This approach allows DBSCAN to:

- Discover clusters of arbitrary shape and size.
- Identify and exclude outliers or noisy data points.
- Avoid the need to predefine the number of clusters.

However, DBSCAN has limitations:

- Sensitive to the choice of ε and `minPts`.
- Struggles with datasets containing varying densities.

7.2.3 Example

In the case of two interlocking half-moon shapes (a common synthetic dataset), DBSCAN successfully separates the shapes using density rather than geometric assumptions. Core points within each moon shape form clusters, border points are appended, and outliers are excluded.

7.2.4 HDBSCAN: Hierarchical DBSCAN

HDBSCAN (Hierarchical DBSCAN) is an extension of DBSCAN that improves upon its weaknesses. It eliminates the need to set ε manually and adapts to varying local densities.

Key features include:

- Builds a hierarchy of clusters by varying the density threshold.
- Selects the most stable clusters via *persistence*, a measure of how long a cluster exists as density thresholds change.
- More robust to noise and better at identifying meaningful clusters of different densities.

7.2.5 How HDBSCAN Works

1. Each point starts as its own cluster.
2. A mutual reachability graph is built, using distance scaled by local density.
3. Clusters are merged in a hierarchical fashion as the density threshold decreases.
4. A condensed cluster tree is produced by pruning low-stability branches.

7.2.6 DBSCAN vs HDBSCAN

Aspect	DBSCAN	HDBSCAN
Input parameters	ε, `minPts`	`min_cluster_size`, `min_samples`
Cluster shapes	Arbitrary	Arbitrary
Density adaptation	Fixed	Variable (adaptive)
Handles noise	Yes	Yes (better)
Requires ε	Yes	No
Scalability	Good	Moderate

7.2.7 Use Case: Canadian Museums

When applied to spatial data such as latitudes and longitudes of Canadian museums, DBSCAN was able to detect natural geographic clusters but struggled in high-density urban areas by lumping many points together. HDBSCAN, with adaptive neighbourhood sizing, successfully identified finer structures and revealed multiple clusters within dense metropolitan regions.

7.3 Dimensionality Reduction

Dimensionality reduction refers to techniques that reduce the number of features in a dataset while preserving essential information. High-dimensional data can be challenging to visualise and computationally expensive to process. These methods aim to project the data into a lower-dimensional space that captures the structure, variance, or similarity relationships inherent in the original data.

Popular dimensionality reduction algorithms include:

- Principal Component Analysis (PCA)
- t-Distributed Stochastic Neighbour Embedding (t-SNE)
- Uniform Manifold Approximation and Projection (UMAP)

7.3.1 Principal Component Analysis (PCA)

Principal Component Analysis is a linear technique that assumes the features in a dataset are linearly correlated. PCA projects the data onto a new coordinate system defined by orthogonal directions called *principal components*, which are ordered by the amount of variance they capture from the original data.

- The first principal component captures the direction of maximum variance.
- Subsequent components are orthogonal and capture decreasing amounts of variance.
- PCA transforms the original features into a reduced set of uncorrelated variables.

PCA is effective when the structure in the data is linear and often used for noise reduction and feature compression.

7.3.2 t-Distributed Stochastic Neighbour Embedding (t-SNE)

t-SNE is a non-linear dimensionality reduction algorithm particularly suited for high-dimensional data such as text and image features.

- It maps the data into two or three dimensions by preserving the local similarity between points.
- Points that are close in the high-dimensional space remain close in the low-dimensional embedding.
- Distant points are given less emphasis in the optimisation.

Despite its ability to reveal hidden clusters, t-SNE has limitations:

- High computational cost and poor scalability.
- Sensitivity to hyperparameters such as perplexity and learning rate.

7.3.3 Uniform Manifold Approximation and Projection (UMAP)

UMAP is a more recent non-linear method that improves upon the limitations of t-SNE by incorporating manifold learning.

- It constructs a graph in the original space assuming that the data lies on a low-dimensional manifold.
- A second graph is constructed in low dimensions to preserve the local and global structure of the original data.
- UMAP generally scales better and produces more meaningful embeddings for large and complex datasets.

7.3.4 Comparison Example

To illustrate the comparative behaviour of these algorithms, consider a synthetic dataset comprising four Gaussian blobs generated in three dimensions. The blobs vary slightly in their standard deviations, resulting in slight overlaps between some clusters.

PCA: Since the blobs are linearly separable, PCA performs well, projecting the clusters into 2D while retaining the primary directions of variance.

t-SNE: The algorithm identifies four well-separated clusters but introduces minor misclassifications, particularly where original clusters overlap. t-SNE is effective at revealing local structure but can distort global relationships.

UMAP: UMAP recovers most of the cluster structure while preserving both local and global features. It outperforms t-SNE in this case by better maintaining the density structure of the original data.

8 Evaluating Models

8.1 Supervised Learning

8.1.1 Classification

Supervised learning evaluation measures how effectively a machine learning model can predict outcomes on unseen data. This process is crucial for assessing a model's generalisation capability. Evaluation occurs during both the training and testing phases, ensuring that the model does not merely memorise the training data but captures patterns that extend to new inputs.

A standard approach is the *train-test split*, where the dataset is divided into two subsets:

- **Training set:** Typically 70–80% of the data, used to train the model.
- **Test set:** The remaining 20–30%, used to evaluate model performance on unseen data.

In classification tasks, the model predicts categorical labels. Evaluation metrics compare these predicted labels to the true labels to assess performance.

Accuracy Accuracy is the proportion of correctly predicted instances:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Confusion Matrix A confusion matrix provides a more detailed breakdown:

- **True Positives (TP):** Correctly predicted positive instances.
- **True Negatives (TN):** Correctly predicted negative instances.
- **False Positives (FP):** Incorrectly predicted as positive.
- **False Negatives (FN):** Incorrectly predicted as negative.

Precision Precision is the ratio of true positives to all predicted positives:

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

This metric is important when the cost of false positives is high (e.g. in recommendation systems).

Recall Recall measures the ability of the model to detect actual positives:

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

It is critical when the cost of false negatives is high (e.g. in medical diagnostics).

F1 Score The F1 score is the harmonic mean of precision and recall:

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

It is useful when precision and recall are equally important.

Example: Iris Classification Using the `KNeighborsClassifier` on the `Iris` dataset, one can evaluate classification performance with a confusion matrix and associated metrics for each class: `setosa`, `versicolor`, and `virginica`. A heat map of the confusion matrix visualises correct and incorrect predictions. In a well-performing model, diagonal entries (correct classifications) will dominate.

Class-Specific Metrics Precision, recall, and F1 scores can be computed per class. A weighted average accounts for class imbalance by weighting each metric according to the number of samples in that class.

8.1.2 Regression

Regression models aim to predict continuous numerical values. However, these predictions are not exact and typically involve some degree of error. Evaluation of regression models is therefore essential to assess how well the model approximates true outcomes.

Let us consider an example: predicting final exam scores based on midterm results. A regression line is fit to the data, and the differences between the actual grades and the predicted values form the model's *errors*. These errors, also called residuals, quantify the deviation between the fitted model and the observed data.

Formally, for a dataset $\{(x_i, y_i)\}_{i=1}^{n}$, with predictions $\hat{y}_i$ from the model, several error-based metrics are used to quantify the quality of fit.

Mean Absolute Error (MAE) The MAE measures the average magnitude of the errors:

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

Mean Squared Error (MSE) The MSE penalises larger errors more heavily by squaring the differences:

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

where p is the number of model parameters.

Root Mean Squared Error (RMSE) The RMSE is the square root of the MSE and is often preferred because it shares the same units as the target variable:

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

Coefficient of Determination (R^2) The R^2 score measures the proportion of variance in the target variable explained by the model:

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

An R^2 value of 1 indicates perfect prediction, while 0 means the model performs no better than simply predicting the mean $\bar{y}$ of the observed data. Negative values imply that the model performs worse than the mean-value predictor.

Explained Variance Explained variance quantifies the amount of variation captured by the model:

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

It is closely related to R^2 and equals 1 for perfect predictions.

Interpreting Metrics

- A lower MAE, MSE, and RMSE indicate smaller prediction errors.
- A higher R^2 or explained variance reflects a better model fit.
- RMSE is sensitive to outliers due to the squaring operation.
- MAE is more robust to outliers.
- R^2 may be misleading for non-linear relationships.

Example: Effect of Transformations Consider fitting a linear model to a target variable following a log-normal distribution. Transformations such as the logarithm or Box-Cox improve linearity. Visually, this causes the data to cluster more closely around the best-fit line. Consequently, error metrics (MAE, MSE, RMSE) decrease, and R^2 increases, reflecting improved model performance.

8.2 Unsupervised Learning

Evaluating unsupervised learning models presents distinct challenges, as there are no predefined labels or ground truths to compare against. Unlike supervised learning, the goal is to uncover hidden structures or patterns within the data. Evaluation focuses on the coherence, separability, and stability of these patterns.

Challenges and Importance Without target labels, unsupervised model evaluation relies heavily on heuristic metrics, domain-specific insights, and qualitative analysis. A good model should group similar points into coherent clusters and preserve the essential structure of the data, even under perturbations. *Stability* refers to a model's ability to produce consistent clusters across variations in the dataset.

Effective evaluation therefore involves a multifaceted approach combining:

- Internal evaluation metrics
- External evaluation metrics (when labels are available)
- Visual inspection and dimensionality reduction
- Domain expertise
- Model stability analysis

Evaluating Clustering Models

Clustering aims to partition data into homogeneous groups. Its evaluation typically relies on either internal or external metrics.

Internal Evaluation Metrics These metrics assess clustering quality based solely on the data and the resulting cluster structure:

- **Silhouette Score:**
$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \in [-1, 1]$$
where $a(i)$ is the average intra-cluster distance and $b(i)$ is the lowest average inter-cluster distance. Higher scores indicate better separation and cohesion.

- **Davies-Bouldin Index (DBI):** Measures average similarity between clusters:
$$\text{DBI} = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)} \right)$$
where σ_i is the average distance of points in cluster i to its centroid c_i, and d is the distance between centroids. Lower values indicate better clustering.

- **Inertia:** Also called within-cluster sum of squares:
$$\text{Inertia} = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$
where μ_i is the centroid of cluster C_i. Lower values imply tighter clusters.

External Evaluation Metrics When ground-truth labels are available, these metrics compare predicted cluster assignments to true labels:

- **Adjusted Rand Index (ARI):** Measures similarity between clustering and ground truth:
$$\text{ARI} \in [-1, 1], \quad 1 = \text{perfect match}, \quad 0 = \text{random}, \quad < 0 = \text{worse than random}$$

- **Normalized Mutual Information (NMI):**
$$\text{NMI}(U, V) = \frac{2 \cdot I(U; V)}{H(U) + H(V)} \in [0, 1]$$
where I is mutual information and H is entropy. Measures shared information between cluster labels and true classes.

- **Fowlkes–Mallows Index (FMI):**
$$\text{FMI} = \sqrt{\frac{TP}{TP + FP} \cdot \frac{TP}{TP + FN}}$$
A geometric mean of precision and recall for clustering results.

Visualization and Dimensionality Reduction Scatter plots of dimensionality-reduced data help qualitatively assess clustering. Techniques include:

- **PCA:** Projects data onto orthogonal principal components. The *explained variance ratio* indicates how much information is retained in each component.
- **t-SNE and UMAP:** Nonlinear methods that preserve local structure. Evaluation includes:
  - *Reconstruction Error:* Measures how well the original data can be recovered.
  - *Neighborhood Preservation:* Assesses retention of local relationships.

Case Study: K-Means Clustering Simulated blob data with varying separability is clustered using K-Means. The following patterns emerge:

- Clear separation results in high silhouette scores (e.g., $s = 0.84$) and low DBI ($= 0.22$).
- Poor separation yields lower silhouette scores (≈ 0.58) and higher DBI.
- Increasing k reduces inertia but may lead to over-segmentation.

9 Model Generalisability

9.1 Model Validation

10 Model Generalisability

10.1 Model Validation

Model validation is a critical component in the development of machine learning models. Its primary goal is to ensure that a model generalizes well to unseen data, rather than merely fitting the training data. This process safeguards against overfitting, particularly during the tuning of hyperparameters, and helps assess how well a model will perform in practical, real-world scenarios.

Motivation

Many machine learning models come with hyperparameters—settings that must be chosen prior to training and which directly affect model performance. If one naively tunes hyperparameters based on performance on the test set, this effectively fits the model to the test data, compromising its role as an unbiased benchmark. This practice, known as *data snooping* or a form of *data leakage*, leads to overfitting and invalidates test set performance as an estimator of real-world generalisation.

Train-Validation-Test Split

To address this, model validation requires a disciplined separation of data:

- **Training Set:** Used to fit the model and update its internal parameters.
- **Validation Set:** A holdout set from the training data used to tune hyperparameters and assess model performance during development.
- **Test Set:** A final, untouched set used solely for evaluating the generalisation ability of the final model.

This three-way split ensures that hyperparameter optimisation remains decoupled from final evaluation, avoiding the pitfalls of data leakage.

Cross-Validation

Using a single validation set can lead to instability—model performance may vary significantly depending on which subset is chosen. Furthermore, in data-constrained environments, reserving a separate validation set may reduce the amount of data available for training, leading to biased estimates of model performance.

K-fold cross-validation provides a principled solution. The data is split into K equally sized folds. For each fold:

1. Train the model on the remaining $K - 1$ folds.
2. Evaluate it on the k^{th} fold (the validation fold).

This process is repeated K times, with each fold serving as the validation set once. The model's performance is then averaged over all K trials. This ensures that:

- Every data point is used for both training and validation.
- Variance from arbitrary splits is reduced.
- Hyperparameter selection becomes more robust.

Typical values for K range from 5 to 10, striking a balance between computational efficiency and robustness.

Stratified Cross-Validation and Skewed Targets

In classification, datasets often exhibit class imbalance—where some classes dominate in frequency. This can bias evaluation. **Stratified cross-validation** preserves class proportions across folds, ensuring a fairer evaluation across all classes.

In regression, the analogous issue arises when the target variable is highly skewed. Many regression models implicitly assume a normally distributed target. In such cases, one can apply transformations to the target variable:

- **Logarithmic Transformation:** Useful when the target exhibits exponential growth.
- **Box-Cox Transformation:** A family of power transformations that can normalise skewed distributions.

These transformations can improve linearity and model fit, thereby enhancing predictive accuracy and stability.

10.2 Regularisation in Regression and Classification

Regularisation is a technique used to prevent overfitting in both regression and classification models. Overfitting occurs when a model learns the noise in the training data rather than the underlying pattern, leading to poor generalisation to unseen data. Regularisation combats this by constraining the model's flexibility, typically through penalising large coefficients in the objective function.

Linear Regression and Overfitting

In ordinary least squares (OLS) linear regression, the model assumes a linear relationship between the features x_i and the target variable y:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = X\boldsymbol{\theta}$$

Here, X is the feature matrix (with a bias column of 1s), and $\boldsymbol{\theta}$ is the vector of model coefficients. The goal is to minimise the mean squared error (MSE):

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

However, in high-dimensional or noisy data, OLS can lead to large coefficient magnitudes, capturing noise and resulting in poor generalisation.

The Regularised Cost Function

To address this, regularisation modifies the cost function by adding a penalty term that constrains the size of the coefficients:

$$\mathcal{L}_{\text{reg}} = \text{MSE} + \lambda \cdot \text{Penalty}(\boldsymbol{\theta})$$

The hyperparameter λ controls the trade-off between fitting the data and keeping coefficients small.

Ridge and Lasso Regression

Two of the most common regularisation methods are:

Ridge Regression (L2 penalty):

$$\mathcal{L}_{\text{ridge}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2$$

The L2 penalty shrinks all coefficients smoothly toward zero. It is especially effective in scenarios where all features contribute a little.

Lasso Regression (L1 penalty):

$$\mathcal{L}_{\text{lasso}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{n} |\theta_j|$$

The L1 penalty can shrink some coefficients exactly to zero, effectively performing *feature selection*. This sparsity makes lasso particularly useful when the number of features is large and only a subset is expected to be informative.

Performance in Different Noise Environments

The effectiveness of each method varies by signal-to-noise ratio (SNR) and the sparsity of the true coefficients:

- **High SNR, sparse coefficients:** Lasso excels by zeroing out irrelevant features while accurately identifying important ones.
- **Low SNR, sparse coefficients:** Lasso remains robust, outperforming ridge and linear regression, which tend to overfit.
- **High SNR, non-sparse coefficients:** All methods perform reasonably well; lasso identifies some zeros, while ridge is slightly more stable.
- **Low SNR, non-sparse coefficients:** Ridge generally outperforms lasso in fitting non-zero coefficients, but lasso still performs well as a feature selector.

Empirical Comparison

Empirical plots comparing test predictions to actual values show that:

- **Lasso regression** tracks actual values closely with predictions concentrated around the ideal diagonal.
- **Ridge regression** performs better than standard linear regression in noisy settings.
- **Linear regression** often overfits, especially in low SNR environments, resulting in erratic coefficient estimates and poor generalisation.

In a moderately noisy setting, lasso achieves significantly lower MSE than ridge and linear regression—up to 30 times smaller. This demonstrates the power of regularisation in producing generalisable models.

Regularisation in Classification

Regularisation principles extend naturally to classification models, such as logistic regression. The cost function, often based on cross-entropy, is augmented by L1 or L2 penalties to prevent the model from overfitting the training data:

$$\mathcal{L}_{\text{logistic-reg}} = \text{CrossEntropy}(\hat{y}, y) + \lambda \cdot \text{Penalty}(\boldsymbol{\theta})$$

This is particularly useful in high-dimensional spaces, such as text classification, where sparsity and noise are common.

10.3 Data Leakage

Data leakage is a critical issue that arises when information from outside the training dataset, typically unavailable during deployment—leaks into the model training process. This leads to artificially inflated model performance during validation and testing, but poor generalisation in real-world applications.

What is Data Leakage?

Data leakage occurs when the training data includes information that would not be available at prediction time. For example, using a feature engineered from the entire dataset (e.g., the average home price) when predicting individual house prices introduces information from the test set into the training process. This scenario mimics having access to the future, which is unrealistic in production.

Data snooping is a specific form of data leakage where information from the test set indirectly influences the training process, such as tuning hyperparameters based on test set performance or feature engineering using future labels.

Consequences of Leakage

When data leakage is present:

- The model appears to perform well on the test set.
- Model evaluation metrics are misleading.
- The deployed model underperforms, failing to replicate evaluation results.

Leakage invalidates the model validation pipeline, making the trained model unreliable.

Common Sources of Data Leakage

- **Global statistics:** Using dataset-level aggregates (e.g., overall mean, standard deviation) during training.
- **Temporal leakage:** Using future data to predict past or present values.
- **Inappropriate feature engineering:** Deriving features from the entire dataset before splitting into training/test sets.
- **Improper cross-validation:** Applying preprocessing steps (e.g., scaling, PCA) before the data is split into folds.

Preventing Data Leakage

To mitigate the risk of leakage:

- Keep training, validation, and test sets strictly separate with no contamination.
- Ensure feature engineering and preprocessing are performed within a pipeline fitted only on the training data.
- Avoid global statistics computed using the full dataset.
- For time-dependent data, use time-aware splits (e.g., `TimeSeriesSplit`) rather than random train-test splits.
- During cross-validation, fit the pipeline separately within each fold to avoid leakage across folds.