# AI Machine Learning Algorithms and Techniques

## Supervised Learning

Supervised Learning is a foundational approach in machine learning that empowers machines to learn from label data and make predictions on unseen data.

### Key principles:

#### Labeled data

The essence lies in labeled data. Each input (***feature***) in the dataset is paired with a corresponding output (***label***), which serves as “ground truth” from which model learns.

#### Learning from examples

Supervised learning works by learning from examples. The model is fed a set of input–output pairs (training data), and it tries to find patterns or relationships in the data that allow it to predict the output for new inputs. The learning process involves adjusting the internal parameters of the model to minimize errors in prediction, a process called "training."

#### Generalization

Generalization refers to the model's ability to perform well on new data, meaning it has learned the underlying patterns of the problem rather than just memorizing the training examples. One of the most important goals of supervised learning is generalization. A model that performs well on the training data but fails on new, unseen data is said to "***overfit***."

### Types of supervised learning problems

#### Classification

In classification tasks, the goal is to assign the input data to one of several predefined categories or classes. The output is a discrete label. Example: spam vs non-spam, dog vs cat.

In classification, algorithms aim to find *decision boundaries* *that separate the different classes*. These boundaries help the model classify new data points into the correct categories.

#### Regression

Regression involves predicting a continuous numerical value based on input data. The output is not categorical but instead a real number. E.g.: Predicting house price based on size, location.

In regression tasks, the model learns *a function that best fits the data*, allowing it to predict continuous outcomes for new data points.

### Common algorithms used in supervised learning

#### Linear regression (for regression)

Linear regression assumes a *linear relationship* between the input features and the output variable. The algorithm fits a line (or hyperplane) to the data that minimizes the difference between the predicted and actual outputs. Linear regression is easy to implement and works well for simple, linear relationships.

#### Logistic regression (for classification)

Logistic regression is a classification algorithm commonly used for binary classification tasks. It estimates the probability that a given input belongs to a particular class (e.g., spam or not spam) using a logistic function. Logistic regression is widely used because it’s easy to interpret and performs well for linearly separable data.

#### Decision trees (for classification and regression)

Decision trees are versatile algorithms used for both classification and regression tasks. They work by splitting the data into subsets based on feature values, creating a tree-like structure where each node represents a decision. Decision trees are easy to understand and interpret, but they can overfit the data if not properly controlled (e.g., through pruning)

#### Support vector machines (SVM) (for classification)

SVMs are powerful classification algorithms that work well for both linear and nonlinear problems. The algorithm tries to find the optimal hyperplane that separates the data points from different classes. SVMs are particularly effective in high-dimensional spaces, making them useful for complex classification tasks.

#### 3.5 k-nearest neighbors (k-NN) (for classification and regression)

k-NN is a simple, instance-based algorithm that classifies a data point based on the majority label of its "k" nearest neighbors. It’s nonparametric, meaning it doesn’t make assumptions about the data distribution. You can use k-NN for both classification and regression, but the algorithm can become computationally expensive as the dataset grows.

#### Random forests (for classification and regression)

Random forests are an ensemble learning method that builds multiple decision trees and combines their predictions to improve accuracy. By averaging the results of multiple trees, random forests reduce the risk of overfitting and increase robustness. They are widely used for both classification and regression tasks due to their high performance and flexibility.

#### Neural networks (for classification and regression)

Neural networks are inspired by the structure of the human brain and consist of layers of interconnected nodes (neurons). They are highly flexible and can model complex, nonlinear relationships between inputs and outputs. Neural networks are especially useful in tasks such as image recognition, natural language processing, and deep learning applications.

### Key steps in building supervised learning models

#### Data collection and preparation

The first steps in any supervised learning task are collecting and preparing the data. This involves gathering labeled data and performing tasks such as cleaning the data (handling missing values, removing outliers), transforming the data (normalization or scaling), and splitting it into training and test sets.

#### Model training

Once the data has been prepared, the next step is training the model. This involves feeding the labeled data into the algorithm, which adjusts its internal parameters to learn the relationship between the input features and the output labels. This process continues until the model has learned a set of rules or patterns that you can use to make predictions.

#### Model evaluation

In many cases, the initial model may not perform as well as expected. To improve the model, you can adjust ***hyperparameters*** (settings that control the learning process). This process is called "model tuning." You can use techniques such as grid search or random search to find the optimal hyperparameters.

#### Deployment and maintenance

Once the model is performing well, you can deploy it into production, where it makes predictions on new data. It’s important to continuously monitor the model’s performance and update it as new data becomes available to ensure it remains accurate.

### Best practices for implementing supervised learning algorithms

#### Data collection and preparation

Quality of data is key

The quality of your data directly impacts the performance of your supervised learning model. Poor or incomplete data can lead to inaccurate predictions, regardless of the algorithm used. Best practices for ensuring high-quality data include:

* **Handling missing data:** Address missing values in your dataset by using techniques such as imputation (replacing missing values with the mean or median) or removing rows/columns with excessive missing data.
* **Removing outliers:** Identify and remove outliers that can skew your model’s predictions. Outliers are extreme values that don't represent the majority of your data.
* **Feature scaling:** Many supervised learning algorithms (such as support vector machines (SVMs) and k-NN) are sensitive to the scale of features. Applying normalization or standardization ensures that all features contribute equally to the model.

Split your data

Dividing your data into distinct sets is critical to avoid overfitting and ensure that your model generalizes well. Typically, the data is split into:

* **Training set:** The subset of the data used to train the model.
* **Validation set:** Used to tune hyperparameters and make adjustments to improve performance.
* **Test set:** A final set used to evaluate the model’s performance on unseen data. This set should not be used during training or tuning.

#### Model selection

Choose the right algorithm

General guidelines:

* For classification tasks: Algorithms such as logistic regression, decision trees, random forests, and SVMs are commonly used. If the data is linearly separable, logistic regression or SVMs might be the best choice. For more complex datasets, random forests or neural networks may perform better.
* For regression tasks: Linear regression is a good starting point for simple problems, while more complex models, such as decision trees or neural networks, may be necessary for capturing nonlinear relationships.

Avoid overfitting

Overfitting occurs when a model learns the noise in the training data rather than the actual underlying patterns, leading to poor generalization on new data. To prevent overfitting:

* **Simplify the model:** Use a simpler algorithm or reduce the complexity of the model (e.g., by limiting the depth of decision trees).
* **Cross-validation:** Use k-fold cross-validation to better assess model performance across different subsets of the data.
* **Regularization:** Apply regularization techniques (such as L1 or L2 regularization) to penalize large coefficients, encouraging the model to find a balance between fitting the data and maintaining simplicity.

#### Hyperparameter tuning

The importance of hyperparameters

Supervised learning algorithms have hyperparameters that control how the model learns. These parameters need to be fine-tuned to optimize model performance. Examples of hyperparameters include:

* **Learning rate:** Controls how quickly the model adjusts its parameters during training
* **Regularization strength:** Determines the amount of penalty applied to model complexity
* **Number of neighbors (for k-NN):** Determines how many nearby data points are considered when making predictions

Hyperparameter tuning techniques

* **Grid search:** A brute-force method where you specify a range of values for each hyperparameter and evaluate all possible combinations.
* **Random search:** Randomly selects hyperparameter combinations from a defined range. This method can be more efficient than a grid search, especially when there are many parameters to tune.
* **Automated hyperparameter tuning:** Tools such as Bayesian optimization or automated machine learning (AutoML) can help you identify optimal hyperparameters without manual intervention.

#### Model evaluation and metrics

Choose the right evaluation metric

The choice of evaluation metric depends on the type of problem you’re solving:

**For classification:** Common metrics include accuracy, precision, recall, F1 score, and ROC-AUC (i.e., Receiver Operating Characteristic Curve, the Area under the Curve). Accuracy is useful for balanced datasets, while precision and recall are more informative when dealing with imbalanced datasets.

**For regression:** Metrics such as mean squared error (MSE), root mean squared error (RMSE), and R-squared are used to evaluate the performance of regression models.

Use cross-validation

Cross-validation helps ensure that your model generalizes well to new data. In k-fold cross-validation, the dataset is split into k parts, and the model is trained k times, each time leaving out one of the k parts as the test set. This process provides a more accurate estimate of the model's true performance by reducing the risk of overfitting or underfitting.

#### Deployment and monitoring

Deploying the model

Once the model has been trained, tuned, and evaluated, it’s ready for deployment. Deployment involves integrating the model into an application or system where it can make predictions on new data. Best practices include:

* **Version control:** Track different versions of the model to ensure you can revert to previous versions if necessary.
* **Containerization:** Use containerization tools such as Docker to package your model, making it easier to deploy across different environments.

Continuous monitoring and maintenance

After deployment, it’s important to continuously monitor the model’s performance, as data distributions may change over time (a phenomenon known as "*data drift*"). This can cause the model’s accuracy to degrade. Regularly retraining the model on new data can help maintain its performance. Additionally, set up alerts to detect significant drops in performance so that corrective action can be taken quickly.

#### Interpretability and explainability

Make models interpretable

In many applications—especially in industries such as healthcare, finance, and law—it’s critical for models to be interpretable. Decision-makers need to understand why a model is making certain predictions. Simpler models, such as decision trees or linear regression, are inherently interpretable, while more complex models, such as neural networks, require explainability tools.

Use explainability tools

For more complex models, tools such as local interpretable model-agnostic explanations (LIME) or SHapley Additive exPlanations (SHAP) can be used to provide insight into how the model arrived at its predictions. These tools help increase trust in the model’s outputs, especially in critical decision-making scenarios.

### Evaluation metrics for supervised learning models

#### Evaluation metrics for classification models

Classification models predict discrete outcomes, such as whether an email is spam or not spam or whether a customer will churn or remain. Below are some key evaluation metrics used to assess the performance of classification models:

##### Accuracy

Accuracy measures the percentage of correct predictions out of all predictions made.

A close up of a text

AI-generated content may be incorrect.

Example: If a model correctly predicts 90 out of 100 instances, its accuracy is 90 percent. However, accuracy may not always be the best metric for imbalanced datasets, in which one class is much more frequent than the other.

##### Precision

Precision measures the percentage of true positive predictions out of all positive predictions that the model makes. It is important in cases in which false positives are costly, such as in medical diagnoses or spam detection.

A close up of a sign

AI-generated content may be incorrect.

Example

In spam detection, precision is the proportion of emails predicted as spam that are actually spam. A high precision value indicates fewer false positives.

##### Recall (sensitivity or true positive rate)

Recall measures the percentage of true positive predictions out of all actual positives. It is important when the cost of missing positive instances is high, such as in disease detection. Recall measures the ability to correctly identify actual positives.

A black text with black text

AI-generated content may be incorrect.

Example

In a cancer detection model, recall is the proportion of actual cancer cases that the model correctly identifies.

##### F1 score

The F1 score is the harmonic mean of precision and recall. It provides a balanced metric when both precision and recall are important, especially for imbalanced datasets.

A black text on a white background

AI-generated content may be incorrect.

Example

A model with high precision but low recall or vice versa will have a lower F1 score, indicating that it is not performing well on both metrics.

##### Confusion matrix

A confusion matrix is a table used to summarize the performance of a classification model. It shows the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

|  |  |  |
| --- | --- | --- |
|  | Predicted positive | Predicted negative |
| Actual positive | True positive (TP)  *Correctly predicted positive cases.*  Example: TP = 70 | False negative (FN)  *Incorrectly predicted negative cases*  Example: FN = 10 |
| Actual negative | False positive (FP)  *Incorrectly predicted positive cases*  Example: FP = 5 | True negative (TN)  *Correctly predicted negative cases.*  Example: TN = 15 |

From this matrix, you can calculate accuracy, precision, recall, and other metrics. It provides a more comprehensive view of model performance than accuracy alone.

Given these definitions, you can calculate the metrics for the above example as follows:

1. Accuracy: [ \text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} ] = 85%
2. Precision: [ \text{Precision} = \frac{TP}{TP + FP} ] = 93.3%
3. Recall (Sensitivity): [ \text{Recall} = \frac{TP}{TP + FN} ] = 87.5%

##### ROC curve and AUC

The receiver operating characteristic (ROC) curve plots the true positive rate (recall) against the false positive rate (FPR) at different threshold levels. The area under the curve (AUC) measures the overall performance of the classifier.

AUC ranges from 0 to 1, where a value closer to 1 indicates a better-performing model.

ROC AUC is particularly useful when you want to evaluate how well a model can distinguish between classes across different thresholds.

#### Evaluation metrics for regression models

Regression models predict continuous values, such as house prices or temperatures. The following metrics are commonly used to evaluate regression models:

##### Mean squared error (MSE)

MSE measures the average squared difference between the predicted and actual values. It is sensitive to large errors because the errors are squared, making it useful for situations in which larger errors are more significant.

A number and symbols of mathematical equations

AI-generated content may be incorrect.

**Steps to calculate MSE:**

1. Find the difference: Subtract the predicted value from the actual value. This gives you the error for each prediction.
2. Square the difference: Multiply the error by itself. This makes sure that all errors are positive and emphasizes larger errors.
3. Average the squared errors: Add up all the squared errors and then divide by the number of predictions. This gives you the average squared error.

Example

In a house price prediction model, if the predicted price is $200,000 and the actual price is $250,000, the mean squared error for that prediction is $2.5 x 109. Here’s the calculation:

Predicted price: Yi = $200,000

Actual price: Y^i = $250,000

Calculation steps:

1. Error : Actual price – predicted price = 250,000 - 200,000 = 50,000
2. Square the error = (50,000) 2 = 2,500,000,000
3. Since this is a single prediction (n = 1), the MSE is 2,500,000,000

##### Root mean squared error

Root mean squared error (RMSE) is the square root of the mean squared error, which brings the error metric back to the same units as the target variable. RMSE is more interpretable because it is in the same unit as the data being predicted.

A mathematical equation with numbers and symbols

AI-generated content may be incorrect.

Example

If the MSE of a model predicting house prices is $625,000,000, then the RMSE will be $25,000, indicating that on average, the prediction of the house prices deviates from the actual price by $25,000.

##### Mean absolute error

Mean absolute error (MAE) measures the average absolute difference between the predicted and actual values. Unlike MSE, it does not square the errors, so it is less sensitive to outliers.



Example

In a weather forecasting model, MAE tells you the average difference between the predicted and actual temperatures.

##### R-squared (coefficient of determination)

R-squared (also known as the coefficient of determination) is a statistical measure that helps you understand how well your model explains the variability of the data.

R-squared explains the proportion of variance in the dependent variable (*the outcome you are trying to predict*) that is predictable from the independent variable(s) (*the predictors or features you are using*). It ranges from 0 to 1, where 1 indicates a perfect fit (usually unrealistic in real-world scenario), and 0 means the model does not explain any of the variance.

A mathematical equations with numbers

AI-generated content may be incorrect.

Example

An R-squared value of 0.9 means that 90 percent of the variance in house prices is explained by the model’s input features. This suggests a strong relationship between the predictors and the outcome.

In summary, R-squared helps you gauge how well your model fits the data and how much of the outcome can be explained by the predictors.

##### Adjusted R-squared

Adjusted R-squared adjusts the R-squared value based on the number of features in the model. It penalizes the addition of irrelevant features, providing a more accurate measure of model performance, especially in cases of overfitting.

A number and a number

AI-generated content may be incorrect.

Where:

* **n** is the number of data points.
* **p** is the number of predictors in the model.

Steps to Calculate Adjusted R-squared:

1. Calculate R-squared using your regression model.
2. Identify the number of observations (n) and the number of predictors (p).
3. Plug these values into the Adjusted R-squared formula to get the adjusted value.

Example

If adding more features to a model decreases the adjusted R-squared, it suggests that the additional features are not improving the model.

#### Choosing the right evaluation metric

Choosing the right evaluation metric depends on the problem you’re solving and the nature of the data. For example:

* **For imbalanced classification problems**: use precision, recall, F1 Score, or ROC AUC instead of accuracy.
* **For regression models**, if large errors are particularly undesirable, consider using RMSE or MSE. If you want a metric that is less sensitive to outliers, use MAE.
* **For complex models**, look at R-squared and adjusted R-Square to answer how well the model explains the variance in the target variable.

#### Applying metrics and cross-validation

Cross-validation ensures that your model's performance is not dependent on a single train-test split, providing a more reliable measure of its generalization.

Cross-validation is introduced to provide a more robust evaluation of the model's performance. You used five-fold cross-validation, which splits the dataset into five parts (folds), trains the model on four folds, and tests it on the remaining fold. This process is repeated for each fold, and the average performance is calculated.

### Feature selection methods: Backward elimination, forward selection, and LASSO

Feature selection is an essential part of building efficient machine learning models. By selecting the most relevant features, you can improve model performance, reduce overfitting, and enhance interpretability.

Three common techniques for feature selection are:

* **Backward elimination:** Improvesmodel performance by removing the least significant features step by step.
* **Forward selection**: incrementally adds significant features to a model by adding the most significant features one by one.
* L**east absolute shrinkage and selection operator (LASSO):** automatically select important features through regularization. It uses regularization to automatically select features by shrinking irrelevant ones to zero.

#### Backward elimination

Backward elimination is a feature selection technique that starts with all the available features and progressively removes the least significant features one by one. The goal is to eliminate features that do not contribute much to the predictive power of a given model.

Steps of backward elimination

1. Fit the model—e.g., linear regression—with all the features in the dataset.
2. Calculate p-values to determine how statistically significant each feature is.
3. Remove the least significant feature—i.e., the feature with the highest p-value.
4. Repeat the process with the remaining features until all remaining features are statistically significant—i.e., below a predefined significance level, typically 0.05.

Advantages

* Straightforward and intuitive.
* Works well when there are many irrelevant features.

Disadvantages

* Can be computationally expensive for large datasets.
* May remove features that are important in combination with others but seem irrelevant when considered individually.

#### Forward selection

Forward selection is the opposite of backward elimination. Instead of starting with all features, forward selection begins with no features and adds them one by one based on their statistical significance and impact on model performance.

Steps of forward selection

1. Start with an empty model: Begin with no features.
2. Add the most significant feature: Add the feature that has the highest correlation with the target variable or provides the most improvement to the model.
3. Refit the model: After each feature is added, refit the model and evaluate the performance, e.g., using adjusted R-squared or another metric.
4. Repeat: Continue adding features until the addition of further features no longer improves the model’s performance.

Advantages

* Useful when there are many features as it builds the model step by step
* Computationally less expensive than backward elimination for very large datasets

Disadvantages

* May include features that only appear significant due to their relationship with other features
* Slower for datasets with a smaller number of features compared to backward elimination

#### LASSO

LASSO is a type of regularization technique that both selects features and shrinks their coefficients. LASSO adds a penalty term—L1 regularization—to the cost function, which drives some feature coefficients to zero, effectively removing them from the model. This makes LASSO useful for automatic feature selection.

How LASSO works:

**L1 regularization**

The LASSO cost function is the ordinary least squares cost function with an added penalty term that is proportional to the absolute value of the feature coefficients. This penalty term shrinks some coefficients to zero.

A black and white symbol

AI-generated content may be incorrect.

*Where:*

* Y1 are the actual target values.
* ŷᵢ are the predicted target values.
* βj are the feature coefficients.
* λ is the regularization parameter that controls the amount of shrinkage.

**Feature selection**

As the regularization parameter λ increases, more feature coefficients are driven to zero. Only the most significant features are left in the model.

Advantages

* Automatically selects features by shrinking irrelevant feature coefficients to zero
* Helps prevent overfitting by penalizing large coefficients
* Works well with high-dimensional datasets where there are many features

Disadvantages

* May remove features that are important in combination but not individually.
* The regularization parameter λ must be carefully tuned.

## Unsupervised Learning

Unsupervised learning is a branch of ML that deals with **unlabeled** data. The model is provided with a dataset that contains input data without corresponding output labels and the model will identify patterns, groupings or hidden structures within it.

**When to use unsupervised learning**

*Exploratory analysis***:** When you have a large, unlabeled dataset and want to discover patterns, unsupervised learning provides valuable insights.

*Data compression:* For high-dimensional data in which training supervised models is computationally expensive, unsupervised learning can reduce dimensionality and streamline model training.

*Pattern recognition:* This is useful when the goal is to find natural groupings within the data, such as clustering users by behavior or identifying outliers for anomaly detection.

*Preprocessing step:* Unsupervised learning is often used to preprocess data before applying supervised learning models, improving accuracy and performance by filtering out irrelevant or redundant features.

### Key principles

#### No labels, only inputs

In unsupervised learning, the data consists only of input variables (X), with no associated output variables (y). The goal is to analyze and learn from the structure of the data without predefined labels. The absence of labels means the algorithm must infer patterns based on the data’s inherent characteristics.

#### Identifying patterns and structures

The primary objective of unsupervised learning is to find hidden patterns, relationships, or groupings in the data. This is useful in scenarios in which manually labeling data is impractical or you want to explore the dataset to understand it better.

#### Data-driven insights

Since unsupervised learning does not rely on labeled outputs, professionals often use it for exploratory data analysis. By revealing structures such as clusters or associations, unsupervised learning helps you understand the underlying dynamics of the dataset, which can later inform supervised models or decision-making processes.

#### Data dimensionality

Many real-world datasets can have thousands of features, making them complex and difficult to analyze. Unsupervised learning techniques such as dimensionality reduction help simplify these datasets by reducing the number of features while preserving important information.

### Common approaches in unsupervised learning

#### Clustering

Clustering is one of the most common and powerful unsupervised learning techniques in ML. The primary goal of clustering is to group data points into clusters such that points within the same group (or cluster) are more similar to each other than to points in other groups. Professionals use clustering in various domains, such as customer segmentation, image processing, and pattern recognition, to uncover hidden structures in data. Clustering is particularly useful when there is no labeled data and the relationships between data points need to be identified without prior knowledge.

**Definition:** Clustering involves grouping similar data points into clusters based on their similarity. It is a foundational method in unsupervised learning.

**Key algorithms**

* *k-means:* This algorithm partitions data into k clusters based on the distance between data points. Each data point is assigned to the cluster with the nearest centroid.
* *Hierarchical clustering:* This builds a tree of clusters by either iteratively merging or splitting clusters based on their proximity.
* *Density-based spatial clustering of applications with noise (DBSCAN):* This algorithm groups data points based on their density and can find clusters of arbitrary shapes, including noise points.

**Applications:** Customer segmentation, social network analysis, image segmentation, and document clustering

**Example:** A retail company can use clustering to segment customers based on purchasing behavior, creating distinct groups such as budget shoppers, frequent buyers, and luxury spenders.

**Evaluating clustering performance**

Unlike supervised learning, clustering does not have a predefined "correct" output, making it more challenging to evaluate. However, you can use several metrics to assess the quality of the clusters:

***Silhouette score:*** This metric measures how similar a data point is to its own cluster compared to other clusters. A higher silhouette score indicates well-separated clusters.

***Elbow method***: This technique determines the optimal number of clusters for k-means. It involves plotting the within-cluster sum of squares (WCSS) and identifying the "elbow" point where adding more clusters no longer significantly reduces the WCSS.

***Davies–Bouldin index:*** This metric measures the average similarity ratio of each cluster to its most similar cluster. A lower value indicates better clustering.

##### k-means clustering

k-means is one of the most widely used clustering algorithms. It works by partitioning data into a predefined number of clusters (denoted by k). Each data point is assigned to the nearest cluster based on the distance from the cluster's centroid (the center of the cluster).

**How it works**

1. Choose the number of clusters (k).
2. Initialize centroids randomly.
3. Assign each data point to the nearest centroid based on distance (usually Euclidean distance).
4. Update the centroids by calculating the mean of the points in each cluster.
5. Repeat the assignment and update the steps until the centroids no longer move or the assignments do not change.

**Advantages**

* Simple and easy to implement
* Works well with large datasets
* Fast and computationally efficient

**Limitations**

* Requires the number of clusters (k) to be specified in advance
* Sensitive to the initial placement of centroids
* Assumes clusters are spherical and equally sized, which may not always be true

**Use cases**

* Customer segmentation: grouping customers based on purchasing behavior
* Image compression: reducing the number of colors in an image by clustering similar colors together

##### Hierarchical clustering

Hierarchical clustering builds a hierarchy of clusters either by merging smaller clusters into larger ones (*agglomerative clustering*) or splitting larger clusters into smaller ones (*divisive clustering*). The result is often visualized as a dendrogram, a tree-like diagram that shows the relationships between clusters.

**How it works (agglomerative)**

1. Treat each data point as its own cluster.
2. Calculate the distance between each pair of clusters.
3. Merge the two closest clusters.
4. Repeat steps 2 and 3 until all points are merged into a single cluster.

**Advantages**

* No need to specify the number of clusters in advance
* Provides a detailed hierarchy of clusters

**Limitations**

* Computationally expensive for large datasets
* Sensitive to outliers

**Use cases**

* Genomics: grouping genes with similar expression patterns
* Document clustering: grouping text documents by topic

##### Density-based spatial clustering of applications with noise (DBSCAN)

DBSCAN is a powerful clustering technique that groups together data points that are close to each other in terms of density and separates outliers. Unlike k-means, DBSCAN does not require the number of clusters to be specified beforehand. Instead, it identifies dense regions of data points and forms clusters based on a distance metric and a minimum number of points.

**How it works**

1. Start with an arbitrary point, and determine whether it is a core point by checking whether there are enough neighboring points within a given radius (epsilon).
2. If the point is a core point, form a cluster around it.
3. Expand the cluster by adding neighboring points that meet the density requirements.
4. Repeat until all points are either assigned to a cluster or marked as outliers.

**Advantages**

* Can identify clusters of arbitrary shapes
* Automatically handles noise (outliers)
* Does not require the number of clusters to be specified in advance

**Limitations**

* Sensitive to the choice of parameters (epsilon and minPts)
* Struggles with datasets with varying density

**Use cases**

* *Anomaly detection*: identifying outliers in network traffic or fraudulent transactions
* *Geospatial data analysis*: grouping locations based on proximity

##### Gaussian mixture models (GMMs)

GMMs are probabilistic models that assume that the data points are generated from a mixture of several Gaussian distributions (normal distributions). Unlike k-means, which assigns points to a single cluster, GMMs assign probabilities to each point, indicating the likelihood that the point belongs to each cluster.

**How it works**

1. Initialize the parameters of the Gaussian distributions (mean, covariance).
2. For each data point, compute the probability that it belongs to each Gaussian distribution.
3. Update the parameters of the Gaussians based on these probabilities.
4. Repeat the process until the model converges.

**Advantages**

* Can model clusters with different shapes and sizes
* Provides soft clustering, where points can belong to multiple clusters with different probabilities

**Limitations**

* Requires specifying the number of clusters
* May converge to local optima if not properly initialized

**Use cases**

* Customer segmentation: assigning probabilities that a customer belongs to multiple segments
* Speech recognition: modeling the probability of different sound patterns.

#### Dimensionality reduction

**Definition:**

In machine learning, many datasets can contain a large number of features or dimensions. While high-dimensional data often holds valuable information, it can be computationally expensive to process, and the presence of many features can lead to overfitting and reduced model performance***. Dimensionality reduction*** is the process of reducing the number of features (dimensions) in a dataset while retaining as much information as possible.

Dimensionality reduction techniques are broadly categorized into two groups:

* *Feature selection*: Selects a subset of relevant features from the dataset
* *Feature extraction:* Transforms the data into a lower-dimensional space, creating new features from the existing ones.

**Key algorithms**

* *Principal component analysis:* This technique transforms the data into a new coordinate system such that the first few principal components (new axes) explain the most variance in the data.
* *t-distributed stochastic neighbor embedding:* This nonlinear dimensionality reduction technique is particularly effective for visualizing high-dimensional data in a lower-dimensional space (such as 2D or 3D).
* *Autoencoders:* These neural networks are designed to learn a compressed representation of input data, making them effective for feature reduction and data reconstruction.

**Applications:** Reducing noise in data, simplifying visualization of high-dimensional datasets, and speeding up training for ML models.

**Example**: In genetics, researchers can use dimensionality reduction techniques to compress large datasets of gene expression data, allowing them to focus on the most significant factors affecting disease outcomes.

##### Principal component analysis (PCA)

**Principal component analysis (PCA)** is one of the most popular dimensionality reduction techniques. It is a linear transformation method that converts the data into a set of uncorrelated components called principal components. Each principal component captures the maximum variance in the dataset, with the first principal component capturing the most variance and each successive component capturing progressively less.

How PCA works:

* Step 1: Standardize the dataset to ensure all features have a mean of 0 and a standard deviation of 1
* Step 2: Compute the covariance matrix to understand the relationship between different features.
* Step 3: Calculate the eigenvectors and eigenvalues of the covariance matrix to identify the principal components.
* Step 4: Sort the principal components based on the magnitude of their eigenvalues and project the data onto the top k principal components, where k is the desired number of dimensions.

**Advantages**

* Reduces computational complexity by lowering the number of dimensions
* Helps visualize high-dimensional data in 2D or 3D
* Reduces redundancy by capturing the most important information in the first few components

**Limitations**

* PCA is a linear method and may not perform well on data with complex, nonlinear relationships.
* The principal components are difficult to interpret since they are linear combinations of the original features.

**Example use case**

PCA is often used in image processing to reduce the dimensionality of large images, such as reducing a 1000-pixel image to a smaller set of principal components that still capture the essential features.

##### t-Distributed stochastic neighbor embedding (t-SNE)

t-Distributed stochastic neighbor embedding (t-SNE) is a nonlinear dimensionality reduction technique primarily used for data visualization. Unlike PCA, which preserves global relationships in the data, t-SNE is designed to preserve local structures, meaning that it groups similar data points close together in a lower-dimensional space.

**Step-by-step guide**

* Step 1: t-SNE constructs a probability distribution over pairs of high-dimensional data points, where similar data points have a high probability of being close to each other.
* Step 2: It then defines a similar probability distribution in a lower-dimensional space and minimizes the difference (Kullback-Leibler divergence) between the two distributions.
* Step 3: The algorithm iteratively adjusts the positions of the data points in the lower-dimensional space to minimize the divergence.

Advantages

* t-SNE is excellent for visualizing complex, high-dimensional data, such as images or text embeddings.
* It preserves local structures in the data, making it effective for cluster analysis and pattern discovery.

Limitations

* t-SNE is computationally expensive and can take longer to process large datasets compared with PCA.
* It is mainly used for visualization and is not suitable for reducing dimensions for predictive modeling.
* The results can vary based on the hyperparameters used, such as perplexity and learning rate.

Example use case

t-SNE is often used in visualizing high-dimensional data such as word embeddings in natural language processing (NLP) or gene expression data in bioinformatics, allowing researchers to spot patterns and clusters that are not immediately apparent.

##### Autoencoders

Autoencoders are a type of neural network used for nonlinear dimensionality reduction. They are part of unsupervised learning and work by learning a compressed, lower-dimensional representation of the input data. The network consists of two main parts:

* **Encoder**: Compresses the input data into a lower-dimensional latent space
* **Decoder**: Reconstructs the original data from the compressed representation

By training the autoencoder to minimize the difference between the input and the reconstructed output, the network learns to identify the most important features in the data and discard noise.

How autoencoders work:

* Step 1: The encoder part of the network reduces the data to a lower-dimensional representation.
* Step 2: The decoder part reconstructs the input from this compressed representation.
* Step 3: The model is trained to minimize the reconstruction error, ensuring that the compressed representation retains important information from the original input.

Advantages

* Autoencoders can handle nonlinear relationships in data, unlike PCA.
* They are flexible and can be used for various data types, including images, time series, and text.
* They can be combined with deep learning techniques to further improve performance.

Limitations:

* Autoencoders require more computational resources and time for training.
* They can be difficult to interpret, as the learned features in the latent space are not easily understood.
* The quality of the dimensionality reduction depends on the architecture of the autoencoder and its training process.

Example use case

Autoencoders are commonly used for compressing image data. For example, a high-resolution image can be reduced to a smaller latent representation, and then the decoder can reconstruct the original image from this compressed version, allowing for efficient storage and transmission

##### Linear discriminant analysis (LDA)

LDA is another technique used for dimensionality reduction, but it differs from PCA in that it is a supervised learning method. LDA seeks to find a linear combination of features that best separates two or more classes. It reduces dimensions by projecting the data onto a lower-dimensional space while maximizing the distance between different classes.

How LDA works:

* Step 1: LDA calculates the within-class and between-class scatter matrices.
* Step 2: It computes the eigenvalues and eigenvectors of these matrices.
* Step 3: The data is then projected onto the directions that maximize the separation between classes.

**Advantages:**

* LDA is ideal for classification problems because it reduces dimensionality while improving class separability.
* It works well with linearly separable data.

**Limitations**

* LDA is not effective for nonlinearly separable data.
* It requires labeled data, making it less suitable for unsupervised tasks.

Example use case

LDA is commonly used in face recognition tasks, where it reduces the dimensionality of the input image data while enhancing class separability for better classification performance.

#### Anomaly detection

**Definition:** Anomaly detection aims to identify data points that deviate significantly from most of the data. These data points are considered anomalies or outliers.

**Key algorithms**

* *Isolation forest*: This tree-based algorithm isolates anomalies by randomly partitioning the data.
* *k-means for outlier detection:* Clusters are identified, and points farthest from any cluster centroid can be flagged as anomalies.
* *Autoencoders for anomaly detection:* Autoencoders can learn normal data patterns and identify anomalies based on high reconstruction errors.

**Applications**: Fraud detection in finance, equipment failure detection in manufacturing, and network intrusion detection in cybersecurity.

**Example**: Banks use anomaly detection algorithms to flag unusual credit card transactions that may indicate fraudulent activity.

#### Association rule learning

**Definition**: This technique identifies relationships between variables in large datasets. Association rules are often used in market basket analysis to discover product combinations that frequently occur together.

**Key algorithms**

* *Apriori*: This algorithm discovers frequent itemsets and builds association rules in a dataset by identifying patterns of co-occurrence.
* *Eclat*: An alternative to Apriori, Eclat uses depth-first search to discover frequent item sets in a dataset.

**Applications**: Retail basket analysis, recommendation systems, and correlation identification between product sales

**Example**: An online retailer might use association rules to identify that customers who buy laptops often purchase laptop cases, leading to more effective product bundling or cross-selling strategies.

### Comparing unsupervised learning approaches

#### Clustering

Clustering is one of the most common unsupervised learning techniques, in which the goal is to partition data into distinct groups, or clusters, such that data points within a cluster are more similar to each other than to those in other clusters.

##### Key clustering algorithms

**The k-means** method partitions the data into a predefined number of clusters. It minimizes the distance between data points and the cluster center (centroid). It works best with spherical, evenly distributed clusters.

**The DBSCAN** algorithm clusters data based on density. It groups points that are close together, identifying outliers or noise that do not belong to any cluster. It is useful for datasets with clusters of varying shapes and densities.

**The Hierarchical clustering** method builds a hierarchy of clusters by either merging smaller clusters into larger ones (agglomerative) or splitting larger clusters into smaller ones (divisive). It’s useful for visualizing the structure of data.

##### Comparison of clustering algorithms

**k-means** works well with evenly distributed clusters but struggles with nonspherical clusters or noise.K-Means partitions the data into a predefined number of clusters by minimizing the distance to the cluster center. It’s efficient, but best suited for spherical clusters and evenly distributed data.

**DBSCAN** handles irregularly shaped clusters and noise but can be sensitive to the choice of parameters such as eps (maximum distance between points) and min\_samples (minimum points in a cluster). DBSCAN shines when dealing with non-spherical clusters or datasets that have noise. It groups data points based on density and can identify outliers—perfect for datasets with more complex structures.

**Hierarchical clustering** offers a flexible approach that allows you to decide on the number of clusters after examining the dendrogram, but it is computationally expensive for large datasets.

##### When to use clustering

* **Customer segmentation:** to group customers based on similar behaviors or characteristics (e.g., spending habits, demographics)
* **Image segmentation:** to identify distinct regions or objects in an image
* **Genomics**: to identify groups of similar genes or proteins.

#### Dimensionality reduction

Dimensionality reduction aims to reduce the number of features in a dataset while preserving as much useful information as possible. This is especially important when working with high-dimensional data, which can be computationally expensive and lead to overfitting.

##### Key dimensionality reduction techniques

* **PCA** is a linear transformation method that reduces the number of features by projecting the data onto a set of orthogonal components (principal components) that capture the maximum variance.

With PCA, you can reduce the number of features while retaining most of the variance in the data. It’s a great tool for simplifying large datasets when the relationships are mostly linear. In genetics research, datasets often contain thousands of variables representing different genetic markers. PCA is used to reduce this complexity, retaining only the most significant genetic factors. This helps researchers identify patterns that may be associated with specific traits or diseases while working with a more manageable dataset.

* **t-SNE** is a nonlinear technique used primarily for data visualization. It reduces the dimensionality of the data while preserving local structure, making it effective for uncovering clusters or patterns. When you need to *visualize and explore non-linear structures*, like clusters or patterns in high-dimensional data, t-SNE is your go-to. It’s designed to preserve local relationships, making it easier to spot hidden clusters. If you're working on image recognition, t-SNE can help you visualize high-dimensional image data by reducing it to a 2D or 3D space. For example, you could use t-SNE to explore how a neural network distinguishes between different categories of images, such as cats and dogs, by visualizing the clusters formed in the lower-dimensional space.
* **Autoencoders** are neural network-based techniques that compress data into a lower-dimensional space and then reconstruct it. Autoencoders are effective for nonlinear dimensionality reduction and can capture complex patterns in the data.

##### Comparison of dimensionality reduction techniques

**PCA** is best suited for linear relationships and global variance preservation. It’s also computationally efficient for large datasets.

**t-SNE** is ideal for nonlinear data and great for visualizing local structures and clusters. However, it’s computationally expensive for large datasets and not ideal for predictive modeling.

**Autoencoders** are suitable for nonlinear data and tasks requiring reconstruction, but they require more computational resources and training time.

##### When to use dimensionality reduction

* **Feature engineering**: to reduce the dimensionality of a high-dimensional dataset before applying an ML model
* **Visualization:** to visualize high-dimensional data in 2D or 3D
* **Noise reduction:** to filter out noise from the data while retaining important features

#### Anomaly detection

Anomaly detection involves identifying rare or unusual data points that differ significantly from the majority. This is useful in scenarios in which identifying outliers is critical, such as fraud detection or fault detection in systems.

##### Key anomaly detection algorithms

**Isolation forest** is an unsupervised learning algorithm that isolates anomalies by randomly partitioning the data. Anomalies are more likely to be isolated quickly, while normal points require more partitions. In a large dataset of credit card transactions, most represent normal, legitimate behavior. However, fraud cases are rare and don't follow typical patterns. Using techniques like Isolation Forest or DBSCAN, the system can identify transactions that deviate from the norm—such as unusually large purchases, transactions made from uncommon locations, or rapid consecutive transactions across different vendors.

**The One-class SVM** is a variant of SVMs designed for anomaly detection. It learns a boundary that separates normal data from anomalies.

**Autoencoders**, when used for anomaly detection, reconstruct the data, and large reconstruction errors indicate anomalies.

##### Comparison of anomaly detection algorithms

**Isolation forest** is fast and effective for large datasets but may not perform well in high-dimensional spaces.

**One-class SVM** is good for datasets with clear boundaries between normal and anomalous data but can be slow for large datasets.

**Autoencoders** are capable of detecting complex anomalies but require extensive training time and computational resources.

##### When to use anomaly detection

* **Fraud detection:** to detect fraudulent transactions in financial datasets
* **Network security**: to identify unusual patterns in network traffic
* **Industrial equipment monitoring:** to detect faults or failures in machinery by identifying abnormal readings

### Choosing the right approach for different datasets

|  |  |  |
| --- | --- | --- |
| **Dataset type** | **Recommended unsupervised learning approach** | **Reason** |
| Customer segmentation | k-means or DBSCAN | To group customers based on similar behaviors |
| High-dimensional data | PCA or t-SNE | To reduce the number of features or visualize the data |
| Anomaly detection | Isolation forest or one-class SVM | To identify rare or unusual data points |
| Image processing | Autoencoders or t-SNE | To reduce noise or extract patterns in images |
| Time series data | Hierarchical clustering or autoencoders | To uncover patterns or detect anomalies in sequences |

## Reinforcement Learning and other approaches

**Reinforcement learning (RL)** is a type of ML where an agent learns to make decisions by interacting with its environment. Instead of being explicitly told what the correct actions are, the agent learns by trial and error, using rewards and penalties to guide its learning. Over time, the agent’s objective is to maximize cumulative rewards through a series of actions. Unlike supervised learning (where the model learns from labeled data) or unsupervised learning (where the model uncovers hidden patterns), RL focuses on how an agent should act in an environment to achieve a specific goal.

### Key concepts in reinforcement learning

RL operates through a feedback loop between the agent and the environment. Here are the core elements that define this interaction:

* **Agent**: The learner or the decision-maker that interacts with the environment. Examples including robots, game-playing AI or self-driving cars.
* **Environment:** The world the agent interacts with, which provides feedback in the form of states and rewards.
* **State**: a decision of the environment at a particular point in time. It provides the agent with information about the current situation.
* **Action:** A choice made by the agent at any given state. The agent chooses an action which leads to a new state.
* **Reward:** the feedback signal received by the agent after taking an action. The goal is to maximize total reward over time.
* **Policy**: the strategy the agent follows in choosing actions based on the current state. A policy can be deterministic (always taking the same action in a given state) or stochastic (taking different actions with certain probabilities).
* **Value function**: the expected reward the agent can accumulate starting from a particular state. The value function helps the agent understand which states are more valuable in the long run.
* **Q-function:** also known as the action-value function – represents the expected return from taking an action in a given state and following a policy afterward.

### Key principles of Reinforcement Learning

#### Trial and error

In RL, the agent learns by trial and error. Initially, the agent may not know which actions are best, so it tries different actions and learns from the outcomes. Over time, it improves its decision-making process based on the feedback it receives.

#### Exploration vs. exploitation

One of the key challenges in RL is balancing exploration and exploitation:

* **Exploration**: the agent tries new actions to gather more information about the environment.
* **Exploitation**: the agent uses the knowledge it has already gained to maximize its reward. An effective agent must explore enough to learn about the environment but also exploit its current knowledge to maximize the cumulative reward.

#### Markov decision process (MDP)

Reinforcement Learning problems are typically modeled as a ***Markov decision process (MDP),*** which assumes that the future state only depends on the current state and the action taken—not on the history of states. An MDP is defined by:

* A set of states.
* A set of actions.
* A reward function.
* A transition function that determines the next state based on the current state and action.

The process is named after Russian mathematician *Andrey Markov* (1856-1922). The Markov property (or "memoryless" property): The future state depends only on the current state and the action taken, not on the sequence of states and actions that preceded it.

*In simpler terms: Given the present, the future is independent of the past.*

#### Cumulative rewards

The goal in Reinforcement Learning is not just to maximize the immediate reward but to maximize the cumulative (long-term) reward. This means the agent needs to consider the future consequences of its actions and not just the immediate payoff.

#### Temporal difference learning

Temporal difference learning is a key principle in Reinforcement Learning, where the agent updates its value estimates based on the difference between successive state values. This allows the agent to improve its estimates of future rewards incrementally, rather than waiting until the end of a sequence of actions.

### Approaches in Reinforcement Learning

#### Value-based method

In value-based methods, the agent tries to learn the value of each state or state-action pair, which helps it make decisions. The most well-known value-based method is **Q-learning.**

Q-learning is an off-policy algorithm that aims to learn the Q-value of each state-action pair, which represents the expected reward of taking an action in a particular state and following the optimal policy afterward. The agent updates its Q-values iteratively using the **Bellman equation**:

A math equation with numbers

AI-generated content may be incorrect.

**Where:**

* ***Q(s, a)*** is the current Q-value for the state-action pair,
* **s** is the current state,
* **a** is the action taken in the current state,
* **r** is the immediate reward received for the current action,
* γ is the discount factor for future rewards,
* **α** is the learning rate,
* **s**′ is the new state, and
* **a′** is the action that maximizes the future reward in the new state.

#### Policy-based methods

In policy-based methods, the agent directly learns the policy without focusing on value functions. These methods are particularly useful in high-dimensional or continuous action spaces.

Policy gradient methods optimize the policy by adjusting the parameters in the direction of higher expected rewards. Instead of estimating value functions, policy gradients directly modify the policy using gradient descent. Gradient descent is an optimization algorithm used to minimize a function by iteratively moving toward the function's minimum. This is often used in complex environments, such as robotics or games.

#### Actor-critic methods

Actor-critic methods combine value-based and policy-based methods. The actor chooses actions based on a learned policy, while the critic evaluates how good the actions are by using a value function. This approach helps balance the exploration and exploitation trade-off and improves the learning process.

#### Deep Reinforcement learning (DRL)

When combined with deep neural networks, RL is referred to as deep reinforcement learning (DRL). DRL allows the agent to handle complex, high-dimensional environments by using deep networks to approximate the value function or policy. Famous examples of DRL include deep Q-networks, which was used to master video games like Atari.

### Applications of Reinforcement learning

Reinforcement Learning is applied across a wide range of industries due to its ability to solve sequential decision-making problems. Some popular applications include:

* Game AI: AI agents in games like chess, Go, and video games use RL to learn strategies and outperform human players.
* Robotics: RL is used to train robots to perform tasks such as navigation, grasping, and manipulation.
* Autonomous vehicles: self-driving cars use RL to learn how to navigate roads safely while optimizing for efficiency.
* Finance: RL is applied in algorithmic trading, portfolio management, and risk management by learning optimal strategies from market data.
* Health care: in personalized treatment plans, such as dosing strategies for medications, RL is used to determine the best actions to take based on patient data.

### Evaluation metrics for reinforcement learning models

Evaluating reinforcement learning models is essential to understanding how well an agent is learning and performing in a given environment. Unlike traditional supervised learning where performance is measured by loss functions or accuracy, reinforcement learning involves understanding both short-term and long-term decision-making.

#### Cumulative reward

**Definition**

Cumulative reward is the total sum of rewards an agent collects during an episode (or across multiple episodes). This metric measures how well an agent performs based on the reward system defined in the environment.

**Why it’s important**

Cumulative reward gives a direct measure of an agent’s overall performance, indicating whether it is improving over time or exploiting suboptimal actions.

**Key consideration**

For environments where future rewards are more important, you can compute cumulative rewards using discounting with a factor γ (discount factor).

**Example**

If an agent receives rewards of +1, –1, +2, and 10 over an episode, its cumulative reward for that episode is 12. Discounting may reduce future rewards by γ, affecting their weight in the final total.

#### Average reward per episode

**Definition**

Average reward per episode is the cumulative reward an agent receives divided by the number of episodes. This metric helps smooth out performance and identify overall trends over time.

**Why it’s important**

It smooths out performance fluctuations and gives a better picture of the agent’s learning progress. Average rewards provide insight into whether the agent is becoming more consistent over time.

**Key consideration**

Early fluctuations can skew average reward measurements, so it is important to assess average rewards over a long series of episodes to see trends.

**Example**

If the agent’s cumulative rewards over 5 episodes are 10, 12, 8, 9, and 11, the average reward per episode is (10 + 12 + 8 + 9 + 11) / 5 = 10.

#### Episode length

**Definition**

Episode length refers to the number of steps the agent takes to complete an episode (e.g., reaching a goal or failing). In some environments, shorter episode lengths may indicate that the agent is learning to reach the goal faster.

**Why it’s important**

A decrease in episode length over time may suggest that the agent is learning to achieve the desired outcome more efficiently. It’s a useful metric in tasks where the goal is to minimize the time or steps necessary to achieve success.

**Key consideration**

Shorter episode lengths aren’t always better if they result from the agent terminating early due to failures.

**Example**

If an agent solves a maze in 50 steps in episode 1, but later reduces the number of steps to 30 in episode 100, this indicates learning progress in minimizing the time to reach the goal.

#### Time to convergence

**Definition**

This metric measures the number of episodes or steps it takes for the agent to reach a stable policy (i.e., stop significantly improving its performance). An agent has reached convergence when its cumulative reward, actions, and performance stabilize.

**Why it’s important**

Time to convergence indicates how quickly an RL agent learns an optimal (or near-optimal) policy. A lower time to convergence is desirable, particularly in environments where training time is costly or computationally expensive.

**Key consideration**

Ensure that the environment is not too easy or too deterministic, as this could result in artificially fast convergence without true learning.

**Example**

In a 10 x 10 grid environment, if the agent's cumulative reward plateaus after 500 episodes and remains stable, we say the agent has converged after 500 episodes.

#### Policy stability

**Definition**

Policy stability measures how often the agent changes its learned policy (i.e., the set of actions it takes in various states) after reaching convergence. It indicates how confident the agent is in its learned actions.

**Why it’s important**

A highly stable policy means that the agent has learned a consistent set of actions that maximize rewards, whereas instability may indicate that the agent is still exploring or that the environment is dynamic.

**Key consideration**

In non-stationary environments (where the environment changes over time), a highly stable policy may not be ideal, as the agent needs to adapt.

**Example**

After convergence, if the agent frequently changes actions in the same state, this may indicate uncertainty or noise in the policy, suggesting the need for further training or policy refinement.

#### Exploration vs. exploitation ratio

**Definition**

This metric measures the balance between exploration (trying new actions to discover their outcomes) and exploitation (choosing known actions that yield high rewards). The calculation often depends on how frequently the agent explores compared to when it exploits.

**Why it’s important**

A well-balanced exploration vs. exploitation ratio ensures that the agent discovers optimal strategies without getting stuck in suboptimal actions.

**Key consideration**

Too much exploration can slow down learning, while too much exploitation can prevent the agent from finding better solutions.

**Example**

In a Q-learning algorithm, if the agent follows an ϵϵ-greedy strategy with ϵ = 0.1, it explores 10 percent of the time and exploits 90 percent of the time. A too-low ϵ might cause the agent to miss potentially better strategies.

#### Success rate

**Definition**

The success rate measures how often an agent completes a task or reaches a goal within a set number of episodes or steps. It’s a ratio of successful episodes to total episodes.

**Why it’s important**

In many tasks, such as games or robotic control, the success rate is the most direct way to measure the agent’s ability to achieve the desired outcome.

**Key consideration**

The use of success rate often occurs in combination with other metrics to provide a more comprehensive view of performance, especially in environments where completing the task quickly is as important as completing it at all.

**Example**

If an agent completes the task in 80 out of 100 episodes, the success rate is 80 percent.

#### Sample efficiency

**Definition**

Sample efficiency refers to how effectively an agent uses its experiences (state-action-reward tuples) to learn. High sample efficiency means that the agent learns well from relatively few episodes.

**Why it’s important**

Sample efficiency is crucial in environments where collecting data or simulating episodes is expensive or time-consuming (e.g., real-world robotics or complex simulations).

**Key consideration**

Algorithms that prioritize sample efficiency tend to converge faster but may require more sophisticated methods such as experience replay or off-policy learning.

**Example**

An agent that reaches an optimal policy after 500 episodes is more sample-efficient than one that requires 10,000 episodes to achieve the same performance.

#### Computational complexity

**Definition**

Computational complexity measures the required time and resources to run an RL algorithm. This metric helps determine whether an algorithm is feasible for large-scale or real-time applications.

**Why it’s important**

In real-world systems, limited computational power or time constraints can affect the choice of RL algorithms. Efficient algorithms are necessary for applications such as autonomous vehicles or robotic systems.

**Key consideration**

Consider both the time complexity (how long the algorithm takes to converge) and space complexity (memory usage for storing value functions or policies).

**Example**

A model-free algorithm such as Q-learning may require more time to converge than a model-based approach but uses fewer resources since it does not need to model the environment explicitly.

## Comparing Machine Learning paradigms

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Supervised Learning** | **Unsupervised Learning** | **Reinforcement Learning (RL)** |
| **Definition** | A type of ML where the model learns from labeled data. Each input in the dataset is associated with the correct output, and the model’s objective is to learn the mapping from inputs to outputs. | Involves training models on data without labeled outputs. The goal is to discover hidden patterns or structures within the data without any explicit feedback on what those patterns should be. | a learning paradigm where an agent interacts with an environment and learns to take actions that maximize cumulative rewards. The agent receives feedback in the form of rewards or penalties after taking actions but does not receive direct supervision on which actions are optimal. |
| **How it works** | ***Input***: A labeled dataset where each input (*feature*) has a corresponding output label.  ***Learning****:* The model minimizes loss function, comparing its prediction to the true label.  ***Output***: a prediction model that can accurately assign labels to unseen inputs. | ***Input***: an unlabeled dataset, where the relationships between data points are not predefined.  ***Learning***: the model tries to identify patterns or group similar data points together.  ***Output***: clusters of similar data points, lower-dimensional representations, or new features. | ***Input***: an environment in which the agent operates, with states, actions, and rewards.  ***Learning***: the agent learns by exploring the environment, taking actions, and receiving feedback in the form of rewards. Over time, the agent adjusts its policy to maximize the long-term reward.  ***Output***: a policy that dictates which action to take in each state to achieve the highest cumulative reward. |
| **Key characteristics** | ***Direct feedback****:* the model receives explicit feedback (the correct output label) for each prediction during training.  ***Common tasks:*** widely used for classification (e.g., image classification, spam detection) and regression (e.g., predicting house prices).  ***Evaluation:*** the model is evaluated using metrics such as accuracy, precision, recall, and mean squared error (MSE). | ***No feedback***: there are no labels or predefined outputs, so the model does not receive feedback during training.  ***Common tasks:*** clustering (e.g., customer segmentation, image grouping) and dimensionality reduction (e.g., principal component analysis [PCA], t-distributed stochastic neighbor embedding [t-SNE]).  ***Evaluation:*** evaluation is more challenging, as there are no ground truth labels. Methods such as silhouette score or visual inspection of clusters are often used. | ***Delayed feedback:*** Unlike supervised learning, the agent does not receive immediate feedback on each individual action. Rewards may be delayed, and the agent must consider the long-term consequences.  ***Trial-and-error:*** the agent learns through exploration (trying new actions) and exploitation (using known actions that yield high rewards).  ***Common tasks:*** used in tasks that require sequential decision-making, such as game playing (e.g., AlphaGo), robotics, and autonomous vehicle navigation.  **Evaluation:** performance is evaluated by metrics such as cumulative reward, time to convergence, and success rate. |

### When to use each learning paradigm

* **Supervised learning:** best used when you have a labeled dataset and the goal is to make accurate predictions (e.g., identifying whether an email is spam or not).
* **Unsupervised learning:** useful when you want to discover patterns or groupings in the data without predefined labels (e.g., segmenting customers based on purchasing behavior).
* **Reinforcement Learning:** ideal for tasks involving sequential decision-making and environments where feedback comes in the form of rewards, often after a series of actions (e.g., training an agent to navigate a maze or play a video game).

## Deep learning and neural network

Neural networks are a foundational element in modern machine learning. Inspired by the structure and function of the human brain, these powerful models consist of interconnected layers of nodes, or neurons, that work together to process and learn from data.

### Key feature

#### Layers

Neural networks are composed of layers: an **input layer, hidden layers** and an **output layer.** Each layer consists of multiple neurons, and the number of neurons can vary depending on the complexity of the task.

* Input layer: receives the input data.
* Hidden layers: where the actual learning takes place. These layers can be shallow (one or two layers) or deep (many layers), giving rise to the term deep learning.
* Output layer: produces the final prediction or classification.

#### Neurons and weights

Neurons are basic units of computation in a neural network. Each neuron receives one or more inputs, multiplies them by assigned weights, sums the results, and passes the value through an **activation function** to produce an output.

**Weights** are learned during training, and their values determine the strength of the connection between neurons.