|  |  |
| --- | --- |
| Supervised Learning | Unsupervised Learning |
| Learn from data labeled with the “right answers” | Find something interesting in unlabeled data. |

|  |
| --- |
| fit() in scikit-learn - the fit() function is used to **train a model** using data. It takes in the input data (features) and the correct answers (labels or targets) and helps the model learn patterns in the data.  For example:  • If you’re teaching a model to predict house prices, you give fit():  • Features: size of the house, number of rooms, location, etc.  • Labels: actual house prices.  After running fit(), the model is ready to make predictions based on what it has learned. |
| Transform() in scikit-learn - the transform() function is used to **change the data** based on the rules learned during training (or fitting).  For example:  • If you’re scaling numbers (like changing them to a range of 0 to 1):  • First, you use fit() to learn the data’s range (minimum and maximum values).  • Then, you use transform() to adjust the numbers to fit into the 0 to 1 range.  The transform() function applies what was learned during fit() to modify the data in a specific way. |
| ex.  from sklearn.preprocessing import StandardScaler  import pandas as pd  # Sample data  data = {'feature1': [1, 2, 3, 4, 5],  'feature2': [10, 20, 30, 40, 50]}  df = pd.DataFrame(data)  # Create a scaler object  scaler = StandardScaler()  # => See next part to learn why we need a scaler object before calling fit() and transform()  # Fit the scaler to the data (learn mean and standard deviation)  scaler.fit(df)  # Transform the data (apply standardization)  df\_scaled = scaler.transform(df)  print(df\_scaled)    **Explanation:**   1. **1. **Import necessary libraries:****   We import StandardScaler for standardization and pandas for handling DataFrames.   1. **2. **Create sample data:****   We create a simple DataFrame with two features.   1. **3. **Create a scaler object:****   We initialize a StandardScaler object, which will be used to standardize our data.   1. **4. **Fit the scaler:****   The fit method calculates the mean and standard deviation for each feature in the DataFrame.   1. **5. **Transform the data:****   The transform method applies the standardization formula (subtract mean, divide by standard deviation) to each feature, resulting in a standardized DataFrame df\_scaled. |
| We need to create a scaler object in the code below because the scaler object holds the specific method or rules used to scale the data. These rules are calculated based on the data we provide. For example, a scaler (like StandardScaler or MinMaxScaler in scikit-learn) might calculate statistics like the mean and standard deviation (for standardization) or the minimum and maximum values (for normalization).  from sklearn.preprocessing import MinMaxScaler  # Create the scaler object  scaler = MinMaxScaler()  # Fit the scaler to the data and transform it  scaled\_data = scaler.fit\_transform(data)  **ability**: Once the scaler object is created, it “remembers” the scaling rules (e.g., min/max values or mean/standard deviation). You can reuse the same scaler object to transform new data (like test data) consistently.  2. **Separation of steps**: Creating the scaler first allows you to split the process into:  • **Fitting**: Learning the data’s properties (e.g., min/max or mean/std).  • **Transforming**: Applying those learned properties to modify the data.  3. **Consistency**: Using the same scaler ensures that both training and testing data are scaled in the same way, which is critical for machine learning models to perform correctly. |
| Here is an example using a different scaler, ColumnTransfromer:  We use ColumnTransformer in scikit-learn to scale specific columns of a dataset, followed by calling fit and transform.  **Problem**  We have a dataset with numerical and categorical features. We want to scale the numerical columns and leave the categorical column untouched.  import pandas as pd  from sklearn.compose import ColumnTransformer from sklearn.preprocessing import MinMaxScaler  # Example dataset  data = pd.DataFrame({  'age': [25, 30, 35, 40],  'income': [40000, 50000, 60000, 70000],  'city': ['New York', 'Chicago', 'San Francisco', 'Boston']  })  # Define the transformer: scale 'age' and 'income'  scaler = ColumnTransformer(  transformers=[  ('num', MinMaxScaler(), ['age', 'income']), # Scale numerical columns  ],  remainder='passthrough' # Leave other columns (like 'city') unchanged  )  # Fit and transform the data  scaled\_data = scaler.fit\_transform(data)  # Convert back to a DataFrame for clarity  scaled\_df = pd.DataFrame(scaled\_data, columns=['age\_scaled', 'income\_scaled', 'city'])  print(scaled\_df)  **Explanation**  1. **ColumnTransformer**: Specifies which columns to scale (age and income) and leaves the rest as-is (city).  2. **MinMaxScaler**: Scales values in the selected columns to a range between 0 and 1.  3. **fit\_transform**: Combines the fit() and transform() steps to first learn the scaling rules (based on the min/max of age and income) and then apply them to the data. |

Deep Learning (DL)   
- refers to a set of techniques for learning from data using neural networks, particularly deep neural networks (i.e., with many layers). Deep learning is a form of supervised or unsupervised learning and focuses on recognizing patterns in data.

|  |
| --- |
| **Softmax activation function** is used for **multiclass classification**, where the task is to classify an input into one of three or more classes. **It** is commonly used in the output layer of neural networks. Softmax transforms the raw output (logits) of a neural network into a probability distribution over the different classes.  **Example of Softmax in Action**  Consider a neural network that outputs three logits for a three-class problem:  • Logit for class 1: z\_1 = 2.0  • Logit for class 2: z\_2 = 1.0  • Logit for class 3: z\_3 = 0.1  We apply the softmax function to these logits:  1. Compute exponentials for each logit:  • e^{2.0} = 7.39  • e^{1.0} = 2.71  • e^{0.1} = 1.11  2. Compute the sum of the exponentials:  7.39 + 2.71 + 1.11 = 11.21  3. Compute the softmax probabilities:  • Probability of class 1: \frac{e^{2.0}}{11.21} = \frac{7.39}{11.21} = 0.66  • Probability of class 2: \frac{e^{1.0}}{11.21} = \frac{2.71}{11.21} = 0.24  • Probability of class 3: \frac{e^{0.1}}{11.21} = \frac{1.11}{11.21} = 0.10  So, the predicted probabilities are [0.66, 0.24, 0.10]. The network is most confident that the input belongs to class 1 with a probability of 0.66. |

|  |
| --- |
| **Sigmoid activation function** is one of the commonly used activation functions in machine learning, particularly in neural networks. It maps input values to a range between 0 and 1, which makes it useful for binary classification problems.  sigma(x) = frac{1}{1 + e^{-x}}  • x is the input to the function (which could be any real-valued number).  • e is the Euler’s number (approximately 2.718).  **Key Properties:**  1. **Range**: The output of the sigmoid function always lies between 0 and 1. This makes it useful when the output needs to represent probabilities.  2. **Smooth Gradient**: The function has a smooth gradient, which allows for effective backpropagation of errors during training.  3. **Non-linearity**: Even though the sigmoid function is non-linear, it allows neural networks to learn complex patterns.  4. **Symmetry**: The sigmoid is symmetric around 0.5, making it centered, but it is not zero-centered, which can be a drawback when training deep networks.  **Use Cases**  • **Binary classification**: In binary classification tasks, the sigmoid is typically used in the output layer to map the output to a probability between 0 and 1. For example, logistic regression uses the sigmoid function to classify data into two categories.  • **Neural Networks**: While sigmoid used to be common as an activation function in hidden layers, it has been largely replaced by functions like ReLU (Rectified Linear Unit) due to some limitations, which I’ll discuss next.  **Limitations:**  1. **Vanishing Gradient Problem**: When the input values are too large or too small, the gradient of the sigmoid function becomes very small (close to zero). This can slow down learning, especially in deep neural networks, because the weights are updated very slowly during backpropagation.  2. **Not Zero-Centered**: Since the output range is between 0 and 1, the function is not zero-centered. This can affect the optimization process, especially when using gradient descent, as updates may zigzag inefficiently in the optimization space.  **Alternatives:**  Due to its limitations, other activation functions like **ReLU** (Rectified Linear Unit), **tanh**, or **Leaky ReLU** are more commonly used in modern neural networks, particularly in hidden layers. However, sigmoid is still used in the output layer for binary classification. |

**Softmax vs. Sigmoid**

While **sigmoid** is used for binary classification (outputting a value between 0 and 1 for a single class), **softmax** is preferred for multiclass classification because it normalizes a set of outputs into a probability distribution over multiple classes.

|  |
| --- |
|  |

|  |
| --- |
| **Decision tree model** is a type of **supervised learning** algorithm.  • **For Classification**: In a classification problem, the decision tree model is trained on data where the goal is to classify the input into predefined categories (classes). For example, in the MNIST dataset, the goal is to classify images into one of 10 digits (0-9).  • **For Regression**: Decision trees can also be used for regression, where the goal is to predict continuous output values. For example, predicting house prices based on features like size, location, and number of rooms. |

c2w2

[Multi-class Classification](https://www.coursera.org/learn/advanced-learning-algorithms/lecture/4u2wC/multiclass) (vid in link)

- Classification with 1 output.

- Each output has more than 2 output labels

[Multi-Label Classification](https://www.coursera.org/learn/advanced-learning-algorithms/lecture/pjIk0/classification-with-multiple-outputs-optional)

- Classification with multiple outputs

- Each output has more than 2 output labels

|  |
| --- |
| **Random Forest  - leverages the power of multiple decision trees to make accurate and robust predictions.** Its ability to handle large datasets, reduce overfitting, and provide feature importance makes it a popular choice in various machine learning applications. |

|  |
| --- |
|  |

|  |
| --- |
| C3w1  Clustering in Unsupervised Learning - **clustering** is a key technique used to group similar data points into clusters, without predefined labels. Since there is no prior knowledge of the correct output, the algorithm identifies inherent patterns or structures in the data.  **Key Concepts of Clustering:**  1. **Cluster**: A collection of data points aggregated based on similarity. Data points within the same cluster are more similar to each other than to those in other clusters.  2. **Distance Metrics**: Clustering often relies on a distance metric (e.g., Euclidean distance, Manhattan distance) to measure the similarity between data points. |

|  |
| --- |
| c3w1  Anomaly Detection in Unsupervised Learning  - **Anomaly detection** is the process of identifying unusual patterns, observations, or data points that deviate significantly from the majority of the data, referred to as “normal” data. These anomalies can be indicative of rare events, such as fraud, equipment failure, or unusual behaviors in various domains like cybersecurity, finance, and health monitoring.  Note:  - img below from select\_threshold func in Exrx2 in Lab2  Recall  - quantifies the proportion of actual anomalies that were correctly identified by the model. |

|  |
| --- |
| C3w2  Recommender systems  - use a variety of machine learning algorithms depending on the type of recommendation they aim to provide. These algorithms can be grouped into three broad categories: **collaborative filtering**, **content-based filtering**, and **hybrid methods**. |

|  |
| --- |
| **Collaborative filtering** is a popular technique used in recommendation systems to predict the preferences or behavior of users based on the preferences of similar users or items. It is widely applied in platforms like Netflix, Amazon, and Spotify to recommend products, movies, or music. The core idea behind collaborative filtering is that users who have agreed on items in the past will likely agree again in the future.  **Types of Collaborative Filtering:**  1. **User-Based Collaborative Filtering**:  • In user-based collaborative filtering, recommendations are made to a user based on the preferences of similar users.  • **How it works**:  1. **Find similar users** (neighbors) based on their rating or interaction history. Similarity is often measured using cosine similarity, Pearson correlation, or other metrics.  2. **Predict ratings** for items that the target user hasn’t rated by averaging the ratings from similar users.  3. Recommend the top-N items with the highest predicted ratings.  • **Example**: If User A and User B have similar tastes in movies, and User B liked a movie that User A hasn’t seen, the system will recommend that movie to User A.  2. **Item-Based Collaborative Filtering**:  • In item-based collaborative filtering, recommendations are made by finding items similar to the ones the user has liked or interacted with.  • **How it works**:  1. **Find similar items** based on users’ ratings or interactions. Again, similarity can be measured using cosine similarity or Pearson correlation.  2. **Predict ratings** for items that the user hasn’t rated based on the ratings of similar items they’ve rated.  3. Recommend items with the highest predicted ratings.  • **Example**: If a user liked Movie A, and Movie B is similar to Movie A based on the ratings of other users, the system will recommend Movie B.  **Advantages of Collaborative Filtering:**  1. **No Domain Knowledge Required**: Unlike content-based filtering, which relies on domain-specific information (e.g., movie genres or product attributes), collaborative filtering only needs user interaction data.  2. **Personalized Recommendations**: It captures nuanced and complex user preferences, enabling highly personalized recommendations.  **Challenges in Collaborative Filtering:**  1. **Cold Start Problem**:  • **User cold start**: Difficult to recommend items for new users who haven’t interacted with the system yet.  • **Item cold start**: Hard to recommend new items that haven’t received any ratings or interactions.  2. **Sparsity**: In many systems, the user-item matrix is extremely sparse (e.g., users rate only a small subset of items). This makes it challenging for collaborative filtering algorithms to find meaningful patterns.  3. **Scalability**: With a large number of users and items, computing similarities and predictions can become computationally expensive.  **Hybrid Systems:**  Many modern recommendation systems combine collaborative filtering with other techniques, such as **content-based filtering**, to overcome its limitations. For instance, when a system has little collaborative data (e.g., new users), it may rely more on content-based methods that use metadata like item descriptions.  **Applications:**  • **E-commerce**: Amazon uses collaborative filtering to recommend products based on user purchase and browsing history.  • **Streaming Services**: Netflix and Spotify recommend movies and music based on the preferences of similar users or items.  • **Social Media**: Facebook and Instagram suggest friends or posts based on collaborative filtering of user interactions.  Collaborative filtering is a powerful approach in recommendation systems due to its ability to leverage the wisdom of crowds and find latent patterns in user preferences without requiring explicit domain knowledge. |

|  |
| --- |
| **Content-based filtering** is a recommendation system technique that makes predictions based on the attributes of the items and the preferences of the user. It uses item features (e.g., genre, keywords, descriptions) and user profiles to recommend items that are similar to ones the user has liked or interacted with before. Unlike collaborative filtering, content-based filtering does not rely on the preferences or behaviors of other users but focuses solely on the characteristics of the items and the user’s past interactions. |

|  |
| --- |
| C3w3  Reinforcement Learning  **- Reinforcement learning (RL)** is a type of machine learning where an agent learns to make decisions by interacting with an environment. The agent takes actions in the environment, receives feedback in the form of rewards, and adjusts its strategy over time to maximize cumulative reward. Unlike supervised learning, where the model learns from labeled examples, reinforcement learning relies on the agent learning through trial and error. |