Contents

[Machine Learning Algorithms 1](#_Toc174634627)

[Supervised Learning Algorithms**:** 1](#_Toc174634628)

[Unsupervised Learning Algorithms**:** 1](#_Toc174634629)

[Semi-Supervised Learning Algorithms**:** 1](#_Toc174634630)

[Reinforcement Learning Algorithms: 1](#_Toc174634631)

[Dimensionality Reduction Algorithms: 1](#_Toc174634632)

[Ensemble Learning Algorithms: 2](#_Toc174634633)

[Machine Learning Algorithms desecration with Example 2](#_Toc174634634)

[Supervised Learning Algorithms: 2](#_Toc174634635)

[1. Linear Regression 2](#_Toc174634636)

[2. Logistic Regression 2](#_Toc174634637)

[3. Decision Trees 2](#_Toc174634638)

[4. Random Forest 2](#_Toc174634639)

[5. Support Vector Machines (SVM) 2](#_Toc174634640)

[6. K-Nearest Neighbors (KNN) 2](#_Toc174634641)

[7. Naive Bayes 2](#_Toc174634642)

[8. Gradient Boosting Machines (e.g., XGBoost, LightGBM) 2](#_Toc174634643)

[9. Neural Networks (for classification and regression) 3](#_Toc174634644)

[Unsupervised Learning Algorithms: 3](#_Toc174634645)

[1. K-Means Clustering 3](#_Toc174634646)

[2. Hierarchical Clustering 3](#_Toc174634647)

[3. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) 3](#_Toc174634648)

[4. Principal Component Analysis (PCA) 3](#_Toc174634649)

[5. Independent Component Analysis (ICA) 3](#_Toc174634650)

[6. t-Distributed Stochastic Neighbor Embedding (t-SNE) 3](#_Toc174634651)

[7. Gaussian Mixture Models (GMM) 3](#_Toc174634652)

[8. Apriori Algorithm (for association rule learning) 3](#_Toc174634653)

[Semi-Supervised Learning Algorithms: 3](#_Toc174634654)

[1. Label Propagation 3](#_Toc174634655)

[2. Self-Training 3](#_Toc174634656)

[3. Co-Training 3](#_Toc174634657)

[Reinforcement Learning Algorithms: 4](#_Toc174634658)

[1. Q-Learning 4](#_Toc174634659)

[2. Deep Q-Networks (DQN) 4](#_Toc174634660)

[3. Policy Gradient Methods 4](#_Toc174634661)

[4. Actor-Critic Methods 4](#_Toc174634662)

[5. Monte Carlo Methods 4](#_Toc174634663)

[Dimensionality Reduction Algorithms: 4](#_Toc174634664)

[1. Principal Component Analysis (PCA) 4](#_Toc174634665)

[2. Linear Discriminant Analysis (LDA) 4](#_Toc174634666)

[3. t-Distributed Stochastic Neighbor Embedding (t-SNE) 4](#_Toc174634667)

[4. Autoencoders 4](#_Toc174634668)

[Ensemble Learning Algorithms: 4](#_Toc174634669)

[1. Bagging 4](#_Toc174634670)

[2. Boosting (e.g., AdaBoost, Gradient Boosting) 4](#_Toc174634671)

[3. Stacking 4](#_Toc174634672)

[4. Voting Classifier 5](#_Toc174634673)

[Syntax & Example of each with Code 5](#_Toc174634674)

[Supervised Learning Algorithms 5](#_Toc174634675)

[Linear Regression 5](#_Toc174634676)

[Logistic Regression 5](#_Toc174634677)

[Decision Trees 5](#_Toc174634678)

[Random Forest 5](#_Toc174634679)

[Support Vector Machines (SVM) 6](#_Toc174634680)

[K-Nearest Neighbors (KNN) 6](#_Toc174634681)

[Naive Bayes 6](#_Toc174634682)

[Gradient Boosting Machines (e.g., XGBoost, LightGBM) 6](#_Toc174634683)

[Neural Networks 7](#_Toc174634684)

[Unsupervised Learning Algorithms 7](#_Toc174634685)

[K-Means Clustering 7](#_Toc174634686)

[Hierarchical Clustering 7](#_Toc174634687)

[DBSCAN (Density-Based Spatial Clustering of Applications with Noise) 7](#_Toc174634688)

[Principal Component Analysis (PCA) 8](#_Toc174634689)

[Independent Component Analysis (ICA) 8](#_Toc174634690)

[t-Distributed Stochastic Neighbor Embedding (t-SNE) 8](#_Toc174634691)

[Gaussian Mixture Models (GMM) 8](#_Toc174634692)

[Apriori Algorithm 8](#_Toc174634693)

[Semi-Supervised Learning Algorithms 8](#_Toc174634694)

[Label Propagation 8](#_Toc174634695)

[Self-Training 9](#_Toc174634696)

[Co-Training 9](#_Toc174634697)

[Reinforcement Learning Algorithms 9](#_Toc174634698)

[Q-Learning 9](#_Toc174634699)

[Deep Q-Networks (DQN) 10](#_Toc174634700)

[Policy Gradient Methods 10](#_Toc174634701)

[Actor-Critic Methods 10](#_Toc174634702)

[Monte Carlo Methods 10](#_Toc174634703)

[Dimensionality Reduction Algorithms 11](#_Toc174634704)

[Principal Component Analysis (PCA) 11](#_Toc174634705)

[Linear Discriminant Analysis (LDA) 11](#_Toc174634706)

[t-Distributed Stochastic Neighbor Embedding (t-SNE) 11](#_Toc174634707)

[Autoencoders 11](#_Toc174634708)

[Ensemble Learning Algorithms 12](#_Toc174634709)

[Bagging 12](#_Toc174634710)

[Boosting (e.g., AdaBoost, Gradient Boosting) 12](#_Toc174634711)

[Stacking 12](#_Toc174634712)

[Voting Classifier 13](#_Toc174634713)

# Machine Learning Algorithms

Supervised Learning Algorithms**:**

1. Linear Regression
2. Logistic Regression
3. Decision Trees
4. Random Forest
5. Support Vector Machines (SVM)
6. K-Nearest Neighbors (KNN)
7. Naive Bayes
8. Gradient Boosting Machines (e.g., XGBoost, LightGBM)
9. Neural Networks

Unsupervised Learning Algorithms**:**

1. K-Means Clustering
2. Hierarchical Clustering
3. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
4. Principal Component Analysis (PCA)
5. Independent Component Analysis (ICA)
6. t-Distributed Stochastic Neighbor Embedding (t-SNE)
7. Gaussian Mixture Models (GMM)
8. Apriori Algorithm

Semi-Supervised Learning Algorithms**:**

1. Label Propagation
2. Self-Training
3. Co-Training

## Reinforcement Learning Algorithms:

1. Q-Learning
2. Deep Q-Networks (DQN)
3. Policy Gradient Methods
4. Actor-Critic Methods
5. Monte Carlo Methods

## Dimensionality Reduction Algorithms:

1. Principal Component Analysis (PCA)
2. Linear Discriminant Analysis (LDA)
3. t-Distributed Stochastic Neighbor Embedding (t-SNE)
4. Autoencoders

## Ensemble Learning Algorithms:

1. Bagging
2. Boosting (e.g., AdaBoost, Gradient Boosting)
3. Stacking
4. Voting Classifier

# Machine Learning Algorithms desecration with Example

## Supervised Learning Algorithms:

1. Linear Regression  
   *Description*: A linear approach for modeling the relationship between a dependent variable and one or more independent variables.  
   *Example*: Predicting house prices based on square footage.
2. Logistic Regression  
   *Description*: Used for binary classification, estimating the probability of a binary outcome based on one or more predictors.  
   *Example*: Determining whether an email is spam or not.
3. Decision Trees  
   *Description*: A tree-like model of decisions and their possible consequences, including chance event outcomes.  
   *Example*: Classifying whether a customer will purchase a product based on their demographic data.
4. Random Forest  
   *Description*: An ensemble method that combines multiple decision trees to improve predictive performance.  
   *Example*: Predicting loan default risk based on borrower information.
5. Support Vector Machines (SVM)  
   *Description*: A classification method that finds the hyperplane which best separates data into different classes.  
   *Example*: Classifying different species of flowers based on petal and sepal measurements.
6. K-Nearest Neighbors (KNN)  
   *Description*: A non-parametric method used for classification and regression by comparing the input with the closest data points.  
   *Example*: Recommending movies to a user based on similar users' ratings.
7. Naive Bayes  
   *Description*: A probabilistic classifier based on applying Bayes' theorem with strong independence assumptions between features.  
   *Example*: Email filtering to classify emails as spam or non-spam.
8. Gradient Boosting Machines (e.g., XGBoost, LightGBM)  
   *Description*: An ensemble technique that builds models sequentially, each correcting the errors of the previous ones.  
   *Example*: Predicting customer churn in telecom.
9. Neural Networks (for classification and regression)  
   *Description*: Computational models inspired by human neural networks, used for a wide range of tasks.  
   *Example*: Image recognition, such as identifying objects in photos.

## Unsupervised Learning Algorithms:

1. K-Means Clustering  
   *Description*: A clustering algorithm that partitions data into k clusters based on feature similarity.  
   *Example*: Customer segmentation based on purchasing behavior.
2. Hierarchical Clustering  
   *Description*: A method of cluster analysis that seeks to build a hierarchy of clusters.  
   *Example*: Organizing documents into a tree-like structure based on content similarity.
3. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)  
   *Description*: A clustering algorithm that groups together points that are closely packed, marking outliers as noise.  
   *Example*: Identifying geographic regions of high earthquake activity.
4. Principal Component Analysis (PCA)  
   *Description*: A dimensionality reduction technique that transforms data into fewer dimensions while retaining most of the variance.  
   *Example*: Reducing the number of features in a dataset before applying a classifier.
5. Independent Component Analysis (ICA)  
   *Description*: A computational technique for separating a multivariate signal into additive, independent components.  
   *Example*: Blind source separation, like separating different audio signals from a mixed recording.
6. t-Distributed Stochastic Neighbor Embedding (t-SNE)  
   *Description*: A technique for dimensionality reduction that is particularly well-suited for visualizing high-dimensional data.  
   *Example*: Visualizing clusters of handwritten digits.
7. Gaussian Mixture Models (GMM)  
   *Description*: A probabilistic model that assumes all data points are generated from a mixture of a finite number of Gaussian distributions.  
   *Example*: Modeling customer purchasing behavior with multiple underlying profiles.
8. Apriori Algorithm (for association rule learning)  
   *Description*: A classic algorithm used for mining frequent itemsets and relevant association rules.  
   *Example*: Market basket analysis to find product purchase correlations.

## Semi-Supervised Learning Algorithms:

1. Label Propagation  
   *Description*: A method that spreads labels through a graph to label previously unlabeled data points.  
   *Example*: Enhancing a small labeled dataset with a large amount of unlabeled data for image classification.
2. Self-Training  
   *Description*: A method where a model is trained on labeled data and then used to label new data, which is then added to the training set.  
   *Example*: Expanding the training set of a language model by self-labeling a large corpus of text.
3. Co-Training  
   *Description*: A method that uses two classifiers to iteratively label new data, each learning from the other’s labeled set.  
   *Example*: Improving web page classification by leveraging different feature sets like text and images.

## Reinforcement Learning Algorithms:

1. Q-Learning  
   *Description*: A model-free reinforcement learning algorithm to learn the value of actions in a given state.  
   *Example*: A robot learning to navigate a maze by maximizing rewards.
2. Deep Q-Networks (DQN)  
   *Description*: An extension of Q-learning using deep neural networks to estimate the Q-values.  
   *Example*: Teaching an AI agent to play video games by learning optimal strategies.
3. Policy Gradient Methods  
   *Description*: Methods that optimize the policy directly by gradient ascent on expected rewards.  
   *Example*: Training a robot to walk by optimizing the policy that dictates its movements.
4. Actor-Critic Methods  
   *Description*: Combines policy gradient methods with value function estimation to reduce variance in policy updates.  
   *Example*: Balancing a cart-pole system by adjusting the pole's angle and cart's position.
5. Monte Carlo Methods  
   *Description*: A class of algorithms that rely on repeated random sampling to obtain numerical results, often used in reinforcement learning.  
   *Example*: Estimating the optimal path in a game environment by simulating many possible outcomes.

## Dimensionality Reduction Algorithms:

1. Principal Component Analysis (PCA)  
   *Description*: A technique that transforms data to a lower-dimensional space while maximizing variance.  
   *Example*: Reducing the complexity of facial recognition data while retaining key features.
2. Linear Discriminant Analysis (LDA)  
   *Description*: A technique used for dimensionality reduction that also maximizes class separability.  
   *Example*: Preprocessing step before applying a classifier to a high-dimensional dataset.
3. t-Distributed Stochastic Neighbor Embedding (t-SNE)  
   *Description*: A method that converts high-dimensional data into a low-dimensional space for visualization.  
   *Example*: Visualizing high-dimensional biological data to reveal clusters.
4. Autoencoders  
   *Description*: A type of neural network used for learning efficient representations of data, typically for the purpose of dimensionality reduction.  
   *Example*: Compressing image data for more efficient storage.

## Ensemble Learning Algorithms:

1. Bagging  
   *Description*: An ensemble method that reduces variance by training multiple models on different subsets of the data and averaging their predictions.  
   *Example*: Using a bagged decision tree ensemble for more stable predictions in a medical diagnosis task.
2. Boosting (e.g., AdaBoost, Gradient Boosting)  
   *Description*: An ensemble technique that combines weak learners sequentially to create a strong learner, with each new model focusing on the errors of the previous ones.  
   *Example*: Boosting algorithms like XGBoost for improving accuracy in customer churn prediction.
3. Stacking  
   *Description*: An ensemble method that combines multiple models (typically of different types) by training a meta-model to make final predictions based on their outputs.  
   *Example*: Stacking a logistic regression model, a decision tree, and a neural network for better performance in a Kaggle competition.
4. Voting Classifier  
   *Description*: An ensemble method that aggregates the predictions of multiple models to make a final prediction based on majority voting or averaging.  
   *Example*: Combining multiple classifiers to improve accuracy in sentiment analysis.

# Syntax & Example of each with Code

## Supervised Learning Algorithms

### Linear Regression

python

Copy code

from sklearn.linear\_model import LinearRegression

# Initialize the linear regression model

model = LinearRegression()

# Fit the model on training data

model.fit(X\_train, y\_train)

# Predict on the test data

predictions = model.predict(X\_test)

### Logistic Regression

python

Copy code

from sklearn.linear\_model import LogisticRegression

# Initialize the logistic regression model

model = LogisticRegression()

# Fit the model on training data

model.fit(X\_train, y\_train)

# Predict on the test data

predictions = model.predict(X\_test)

### Decision Trees

python

Copy code

from sklearn.tree import DecisionTreeClassifier

# Initialize the decision tree classifier

model = DecisionTreeClassifier()

# Fit the model on training data

model.fit(X\_train, y\_train)

# Predict on the test data

predictions = model.predict(X\_test)

### Random Forest

python

Copy code

from sklearn.ensemble import RandomForestClassifier

# Initialize the random forest classifier

model = RandomForestClassifier()

# Fit the model on training data

model.fit(X\_train, y\_train)

# Predict on the test data

predictions = model.predict(X\_test)

### Support Vector Machines (SVM)

python

Copy code

from sklearn.svm import SVC

# Initialize the support vector classifier

model = SVC()

# Fit the model on training data

model.fit(X\_train, y\_train)

# Predict on the test data

predictions = model.predict(X\_test)

### K-Nearest Neighbors (KNN)

python

Copy code

from sklearn.neighbors import KNeighborsClassifier

# Initialize the KNN classifier with 3 neighbors

model = KNeighborsClassifier(n\_neighbors=3)

# Fit the model on training data

model.fit(X\_train, y\_train)

# Predict on the test data

predictions = model.predict(X\_test)

### Naive Bayes

python

Copy code

from sklearn.naive\_bayes import GaussianNB

# Initialize the Gaussian Naive Bayes classifier

model = GaussianNB()

# Fit the model on training data

model.fit(X\_train, y\_train)

# Predict on the test data

predictions = model.predict(X\_test)

### Gradient Boosting Machines (e.g., XGBoost, LightGBM)

python

Copy code

from xgboost import XGBClassifier

# Initialize the XGBoost classifier

model = XGBClassifier()

# Fit the model on training data

model.fit(X\_train, y\_train)

# Predict on the test data

predictions = model.predict(X\_test)

### Neural Networks

python

Copy code

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Initialize a sequential neural network model

model = Sequential()

# Add layers to the model

model.add(Dense(64, input\_dim=X\_train.shape[1], activation='relu')) # Input layer + first hidden layer

model.add(Dense(1, activation='sigmoid')) # Output layer

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy')

# Train the model on training data

model.fit(X\_train, y\_train, epochs=10, batch\_size=32)

# Predict on the test data

predictions = model.predict(X\_test)

## Unsupervised Learning Algorithms

### K-Means Clustering

python

Copy code

from sklearn.cluster import KMeans

# Initialize the K-Means model with 3 clusters

model = KMeans(n\_clusters=3)

# Fit the model on the data

model.fit(X)

# Predict the cluster labels for the data

labels = model.predict(X)

### Hierarchical Clustering

python

Copy code

from scipy.cluster.hierarchy import dendrogram, linkage

# Perform hierarchical/agglomerative clustering

Z = linkage(X, method='ward')

# Generate a dendrogram to visualize the clusters

dendrogram(Z)

### DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

python

Copy code

from sklearn.cluster import DBSCAN

# Initialize the DBSCAN model with specified parameters

model = DBSCAN(eps=0.5, min\_samples=5)

# Fit the model and predict the cluster labels

labels = model.fit\_predict(X)

### Principal Component Analysis (PCA)

python

Copy code

from sklearn.decomposition import PCA

# Initialize the PCA model to reduce data to 2 components

pca = PCA(n\_components=2)

# Fit and transform the data

X\_reduced = pca.fit\_transform(X)

### Independent Component Analysis (ICA)

python

Copy code

from sklearn.decomposition import FastICA

# Initialize the ICA model to reduce data to 2 components

ica = FastICA(n\_components=2)

# Fit and transform the data

X\_reduced = ica.fit\_transform(X)

### t-Distributed Stochastic Neighbor Embedding (t-SNE)

python

Copy code

from sklearn.manifold import TSNE

# Initialize the t-SNE model to reduce data to 2 dimensions

tsne = TSNE(n\_components=2)

# Fit and transform the data

X\_reduced = tsne.fit\_transform(X)

### Gaussian Mixture Models (GMM)

python

Copy code

from sklearn.mixture import GaussianMixture

# Initialize the GMM model with 3 components

gmm = GaussianMixture(n\_components=3)

# Fit the model on the data

gmm.fit(X)

# Predict the cluster labels

labels = gmm.predict(X)

### Apriori Algorithm

python

Copy code

from mlxtend.frequent\_patterns import apriori, association\_rules

# Generate frequent itemsets with minimum support of 0.1

frequent\_itemsets = apriori(df, min\_support=0.1, use\_colnames=True)

# Generate association rules from the frequent itemsets

rules = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1)

## Semi-Supervised Learning Algorithms

### Label Propagation

python

Copy code

from sklearn.semi\_supervised import LabelPropagation

# Initialize the label propagation model

model = LabelPropagation()

# Fit the model on training data

model.fit(X\_train, y\_train)

# Predict the labels for the test data

predictions = model.predict(X\_test)

### Self-Training

python

Copy code

from sklearn.semi\_supervised import SelfTrainingClassifier

from sklearn.linear\_model import LogisticRegression

# Initialize the base classifier

base\_classifier = LogisticRegression()

# Wrap the base classifier in a self-training model

model = SelfTrainingClassifier(base\_classifier)

# Fit the model on training data

model.fit(X\_train, y\_train)

# Predict the labels for the test data

predictions = model.predict(X\_test)

### Co-Training

python

Copy code

# Co-Training is not directly supported by sklearn; this is a conceptual example

from sklearn.linear\_model import LogisticRegression

# Assume X1 and X2 are two different views (feature sets) of the data

# Initialize two classifiers for the two views

model1 = LogisticRegression()

model2 = LogisticRegression()

# Fit the models on their respective views

model1.fit(X1\_train, y\_train)

model2.fit(X2\_train, y\_train)

# Predict the labels using both models

predictions1 = model1.predict(X1\_test)

predictions2 = model2.predict(X2\_test)

## Reinforcement Learning Algorithms

### Q-Learning

python

Copy code

import numpy as np

# Initialize Q-table with zeros

Q = np.zeros([state\_size, action\_size])

# Update rule for Q-Learning

# Assume reward and next\_state logic is implemented

Q[state, action] = Q[state, action] + alpha \* (reward + gamma \* np.max(Q[next\_state, :]) - Q[state, action])

### Deep Q-Networks (DQN)

python

Copy code

import tensorflow as tf

# Initialize a neural network model for Q-Learning

model = tf.keras.models.Sequential([

tf.keras.layers.Dense(24, activation='relu'), # Hidden layer 1

tf.keras.layers.Dense(24, activation='relu'), # Hidden layer 2

tf.keras.layers.Dense(action\_size, activation='linear') # Output layer for Q-values

])

# Compile the model with Adam optimizer and MSE loss

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.001), loss='mse')

### Policy Gradient Methods

python

Copy code

import tensorflow as tf

# Initialize a neural network model for policy gradients

model = tf.keras.models.Sequential([

tf.keras.layers.Dense(24, activation='relu'), # Hidden layer

tf.keras.layers.Dense(action\_size, activation='softmax') # Output layer for action probabilities

])

# Compile the model with Adam optimizer and categorical crossentropy loss

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.001), loss='categorical\_crossentropy')

### Actor-Critic Methods

python

Copy code

import tensorflow as tf

# Initialize the actor model for action selection

actor = tf.keras.models.Sequential([

tf.keras.layers.Dense(24, activation='relu'), # Hidden layer

tf.keras.layers.Dense(action\_size, activation='softmax') # Output layer for action probabilities

])

# Initialize the critic model for state-value estimation

critic = tf.keras.models.Sequential([

tf.keras.layers.Dense(24, activation='relu'), # Hidden layer

tf.keras.layers.Dense(1, activation='linear') # Output layer for state value

])

### Monte Carlo Methods

python

Copy code

import numpy as np

# Initialize return sums and counts for each state-action pair

returns\_sum = np.zeros([state\_size, action\_size])

returns\_count = np.zeros([state\_size, action\_size])

# Initialize Q-table with zeros

Q = np.zeros([state\_size, action\_size])

# For each episode:

for each episode:

# For each step in episode:

for each step in episode:

# Update the returns sum and count

returns\_sum[state, action] += reward

returns\_count[state, action] += 1

# Update the Q-value

Q[state, action] = returns\_sum[state, action] / returns\_count[state, action]

## Dimensionality Reduction Algorithms

### Principal Component Analysis (PCA)

python

Copy code

from sklearn.decomposition import PCA

# Initialize the PCA model to reduce data to 2 components

pca = PCA(n\_components=2)

# Fit and transform the data

X\_reduced = pca.fit\_transform(X)

### Linear Discriminant Analysis (LDA)

python

Copy code

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

# Initialize the LDA model to reduce data to 2 components

lda = LinearDiscriminantAnalysis(n\_components=2)

# Fit and transform the data (requires labels)

X\_reduced = lda.fit\_transform(X, y)

### t-Distributed Stochastic Neighbor Embedding (t-SNE)

python

Copy code

from sklearn.manifold import TSNE

# Initialize the t-SNE model to reduce data to 2 dimensions

tsne = TSNE(n\_components=2)

# Fit and transform the data

X\_reduced = tsne.fit\_transform(X)

### Autoencoders

python

Copy code

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense

# Define the input dimension

input\_dim = X\_train.shape[1]

# Define the input layer

input\_layer = Input(shape=(input\_dim,))

# Define the encoding layers

encoded = Dense(64, activation='relu')(input\_layer)

encoded = Dense(32, activation='relu')(encoded)

# Define the decoding layers

decoded = Dense(64, activation='relu')(encoded)

decoded = Dense(input\_dim, activation='sigmoid')(decoded)

# Build the autoencoder model

autoencoder = Model(inputs=input\_layer, outputs=decoded)

# Compile the autoencoder model

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

# Train the autoencoder model

autoencoder.fit(X\_train, X\_train, epochs=50, batch\_size=256, shuffle=True)

## Ensemble Learning Algorithms

### Bagging

python

Copy code

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

# Initialize the base estimator (e.g., Decision Tree)

base\_estimator = DecisionTreeClassifier()

# Initialize the Bagging classifier with the base estimator

model = BaggingClassifier(base\_estimator=base\_estimator, n\_estimators=10)

# Fit the Bagging model on training data

model.fit(X\_train, y\_train)

# Predict on the test data

predictions = model.predict(X\_test)

### Boosting (e.g., AdaBoost, Gradient Boosting)

python

Copy code

from sklearn.ensemble import AdaBoostClassifier

# Initialize the AdaBoost classifier

model = AdaBoostClassifier(n\_estimators=50)

# Fit the AdaBoost model on training data

model.fit(X\_train, y\_train)

# Predict on the test data

predictions = model.predict(X\_test)

### Stacking

python

Copy code

from sklearn.ensemble import StackingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

# Define base classifiers

base\_classifiers = [

('dt', DecisionTreeClassifier()),

('svc', SVC())

]

# Define the meta-classifier (e.g., Logistic Regression)

meta\_classifier = LogisticRegression()

# Initialize the Stacking classifier

model = StackingClassifier(estimators=base\_classifiers, final\_estimator=meta\_classifier)

# Fit the Stacking model on training data

model.fit(X\_train, y\_train)

# Predict on the test data

predictions = model.predict(X\_test)

### Voting Classifier

python

Copy code

from sklearn.ensemble import VotingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

# Initialize individual classifiers

clf1 = LogisticRegression()

clf2 = DecisionTreeClassifier()

clf3 = SVC(probability=True)

# Initialize the Voting classifier (soft voting)

model = VotingClassifier(estimators=[

('lr', clf1),

('dt', clf2),

('svc', clf3)

], voting='soft')

# Fit the Voting model on training data

model.fit(X\_train, y\_train)

# Predict on the test data

predictions = model.predict(X\_test)