Common Supervised Algorithms

Classification algorithms are employed when the output variable is categorical. The objective is to assign input data to one of several predefined classes or categories.

- **Purpose:** To categorise data into predefined classes.
- **Output:** Discrete values (e.g., "spam" or "not spam").

Common Classification Algorithms:

1. **Logistic Regression:** Despite its name, logistic regression is used for binary classification tasks (yes/no decisions). It models the probability that a given input belongs to a particular category.
2. **Decision Trees:** These algorithms use a tree structure to make decisions by asking a series of if-else questions, leading to a final decision at the leaves of the tree.
3. **Random Forest:** An ensemble of decision trees that enhances accuracy by aggregating the results of multiple trees, thereby reducing overfitting and improving generalisation.
4. **Support Vector Machines (SVM):** SVM finds the optimal hyperplane that separates classes in high-dimensional space, maximising the margin between different classes.
5. **k-Nearest Neighbours (k-NN):** This algorithm classifies data points based on the majority class of their k closest neighbours in the feature space.
6. **Naive Bayes:** A probabilistic classifier based on Bayes' theorem, assuming independence among features. It's commonly used in text classification tasks.
7. **Neural Networks:** These deep learning models can handle complex classification tasks, including image and speech recognition, by learning hierarchical feature representations.

Regression Algorithms: Regression algorithms are utilised when the output variable is continuous. The goal is to predict a numeric value based on input data.

- **Purpose:** To predict continuous numerical values.
- **Output:** Continuous values (e.g., house prices, stock market prices).

Common Regression Algorithms:

1. **Linear Regression:** Used for predicting continuous outcomes based on linear relationships between the input features and the target variable.
2. **Decision Trees:** Like in classification, decision trees can also be used for regression tasks by predicting continuous values at the leaves based on the input features.
3. **Random Forest:** An ensemble technique that combines multiple regression trees to improve prediction accuracy and reduce variance.
4. **Support Vector Regression (SVR):** An extension of SVM that predicts continuous outcomes while maintaining robustness against outliers.
5. **k-Nearest Neighbours Regression (k-NN):** Similar to k-NN for classification, this algorithm predicts a value based on the average (or weighted average) of the values of its k nearest neighbours.
6. **Naive Bayes:** While primarily a classification technique, some variations can be adapted for regression tasks based on continuous distributions.
7. **Neural Networks:** Capable of modelling complex relationships for regression tasks, neural networks can predict continuous values based on high-dimensional input data.

Here are small code examples for Linear Regression and Logistic Regression with explanations:

1. Linear Regression Example (Predicting house prices):

This code uses linear regression to predict house prices based on the size of the house (in square feet). It trains the model on a dataset, makes predictions on the test set, and evaluates the model's performance using the mean squared error.

```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Example data (size of the house in square feet and
corresponding price)
X = [[800], [1000], [1200], [1500], [2000]]  # Features
(house sizes)
y = [200000, 250000, 300000, 350000, 500000]  # Target
(house prices)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialise the Linear Regression model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Predict the prices for the test set
y_pred = model.predict(X_test)

# Print the test set house sizes and corresponding
predicted prices with unit Rs
print("Test Set House Sizes and Predicted Prices (in
Rs):")
for size, price in zip(X_test, y_pred):
    print(f"House Size: {size[0]} sq ft, Predicted Price:
Rs {price:,.2f}")

# Evaluate the model using mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f"\nMean Squared Error: {mse}")X
```

Output:

```
Test Set House Sizes and Predicted Prices (in Rs):
House Size: 1000 sq ft, Predicted Price: Rs 245,276.87

Mean Squared Error: 22307928.99659401
```

Explanation:

- **Importing necessary libraries**
  - **train_test_split:** This function from sklearn.model_selection is used to split the dataset into training and testing sets. The training set is used to train the model, and the test set is used to evaluate the model's performance.
  - **LinearRegression:** This is a class from sklearn.linear_model that creates a Linear Regression model. Linear regression is a supervised learning algorithm used to predict a continuous target variable based on the input features.
  - **mean_squared_error:** This function from sklearn.metrics is used to calculate the Mean Squared Error (MSE), a common metric to evaluate the performance of regression models. It measures the average squared difference between actual and predicted values.
- **Defining the data**
  - **X:** A list of lists where each inner list represents the size of a house (in square feet). This is the feature used to predict the house price.
  - **y:** A list of corresponding house prices (in Rupees). This is the target variable that the model will predict based on the features in X.
- **Splitting the data into training and testing sets**
  - **train_test_split():** This function splits the dataset into training and **testing sets. Here:**
    - **test_size=0.2:** 20% of the data will be used for testing, and 80% will be used for training.
    - **random_state=42:** This ensures that the split is reproducible, meaning the same data will be split every time the code is run.
  - **X_train and y_train:** These variables store the training data, which will be used to train the model.
  - **X_test and y_test:** These variables store the test data, which will be used to evaluate the model.

- **Initialising the Linear Regression model**
  - This creates an instance of the LinearRegression class. The model will learn the relationship between the house sizes (X_train) and the corresponding prices (y_train).
- **Training the model**
  - **fit():** This method trains the model using the training data (X_train and y_train). The model learns the linear relationship between house sizes and prices during this process.
- **Predicting the house prices for the test set**
  - **predict():** This method uses the trained model to predict house prices for the test set (X_test). The result, y_pred, contains the predicted prices based on the house sizes in X_test.
- **Printing the house sizes and predicted prices**
  - This section prints each house size from the test set along with its corresponding predicted price.
  - **zip(X_test, y_pred):** This combines the house sizes and predicted prices so that they can be printed together.
  - **size[0]:** Extracts the house size (since X_test is a list of lists).
  - **Rs {price:,.2f}:** Formats the predicted price to include commas for readability and rounds it to two decimal places, with the unit "Rs."
- **Evaluating the model using Mean Squared Error (MSE)**
  - **mean_squared_error():** This function calculates the Mean Squared Error between the actual house prices (y_test) and the predicted prices (y_pred). The MSE provides a quantitative measure of how well the model performed; the lower the MSE, the better the model's predictions.
  - The MSE is printed, indicating how far the predictions are, on average, from the actual prices.

2. Logistic Regression Example (Predicting disease presence):

This code demonstrates how to use logistic regression for a simple binary classification problem (disease/no disease) using health indicators. The model is trained on a dataset, and the performance is evaluated using the accuracy score, while the predictions for the test set are printed.

```python
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Example data (patient's health indicators like blood
pressure, cholesterol levels)
X = [[120, 85], [140, 90], [160, 100], [180, 110], [200,
115]]  # Features (health data)
y = [0, 0, 1, 1, 1]  # Target (0 = No disease, 1 =
Disease present)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the Logistic Regression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Predict disease presence for the test set
y_pred = model.predict(X_test)

# Print the test set health data and corresponding
disease predictions
print("Test Set Health Data and Predicted Disease
Presence:")
for data, prediction in zip(X_test, y_pred):
    health_data = f"Blood Pressure: {data[0]},
Cholesterol: {data[1]}"
    disease_status = "Disease Present" if prediction == 1
else "No Disease"
    print(f"{health_data} => Predicted:
{disease_status}")

# Evaluate the model using accuracy score
accuracy = accuracy_score(y_test, y_pred)
        print(f"\nAccuracy: {accuracy * 100:.2f}%")
```

Output:

```
Test Set Health Data and Predicted Disease Presence:
Blood Pressure: 140, Cholesterol: 90 => Predicted: No Disease

Accuracy: 100.00%
```

Detailed Explanation of the Code:

- **Import Necessary Libraries:**
  - **train_test_split:** Splits the dataset into training and testing sets.
  - **LogisticRegression:** A machine learning algorithm used for binary classification problems.
  - **accuracy_score:** Used to evaluate the model's performance by calculating the accuracy of predictions.
- **Example Data:**
  - **X:** Features representing health indicators (blood pressure and cholesterol levels).
  - **y:** The target variable indicating disease presence (0 = No disease, 1 = Disease present).
- **Split the Data:**
  - The dataset is split into training (80%) and testing (20%) sets. The random_state=42 ensures that the split is reproducible.
- **Initialise the Logistic Regression Model:**
  - Initialises the logistic regression model, which will be trained to classify whether a patient has a disease based on their health indicators.
- **Train the Model:**
  - The model is trained using the training set (X_train, y_train). The model learns the relationship between health indicators and disease presence.
- **Predict Disease Presence:**
  - The model makes predictions on the unseen test data (X_test) and stores the predicted values in y_pred.
- **Print Test Set and Predictions:**
  - This loop goes through each test set data point and prints the health indicators (blood pressure and cholesterol) along with the predicted disease status (0 or 1).
- **Evaluate the Model:**
  - The accuracy of the model is calculated by comparing the predicted values (y_pred) with the actual values (y_test). It is then printed as a percentage.