Logistic Regression

**What is Feature Matrix (X) and Target Vector (y):**

1. **Feature Matrix (X)**:
   * **Definition**: The feature matrix, often denoted by X, is a collection of input variables (features) that are used to make predictions. Each row represents a sample, and each column represents a feature.
   * **Example**: In a dataset predicting house prices, the features could include the number of bedrooms, square footage, location, etc.
   * **Shape**: If there are nn samples and mm features, the shape of X is (n,m).
   * **In the provided code**: X is generated by datasets.make\_regression with 100 samples and 1 feature.
2. **Target Vector (y)**:
   * **Definition**: The target vector, often denoted by y, is the collection of output values (labels) that the model is trying to predict.
   * **Example**: In the house prices dataset, y would be the actual prices of the houses.
   * **Shape**: If there are nn samples, the shape of y is (n,)(n,).
   * **In the provided code**: y is the target values corresponding to each sample in X.

**Why Split the Data?**

Splitting the data into training and testing sets is crucial for building and evaluating a machine learning model. Here's why:

1. **Training the Model**:
   * **Purpose**: The training data (X\_train, y\_train) is used to fit the model. The model learns the relationship between features and target values by adjusting its parameters to minimize the error.
   * **Process**: The model iterates over the training data, applying optimization algorithms to learn the best mapping from X to y.
2. **Testing the Model**:
   * **Purpose**: The testing data (X\_test, y\_test) is used to evaluate the model's performance on unseen data. This helps in assessing the model's generalization ability.
   * **Importance**: Without a separate testing set, it's difficult to know if the model has learned the underlying patterns or simply memorized the training data (overfitting).

Training and testing data are essential components of building and evaluating machine learning models. Here are the key reasons for splitting data into training and testing sets:

**1. Model Training:**

* **Training Data**: This subset of the data is used to train the machine learning model. During training, the model learns the underlying patterns, relationships, and features from the data. The goal is to optimize the model parameters so that it can make accurate predictions.
* **Purpose**: To provide the model with enough examples to learn the mapping from input features (X) to output labels (y).

**2. Model Evaluation:**

* **Testing Data**: This subset of the data is used to evaluate the model's performance. The testing data should be independent of the training data to ensure that the evaluation reflects the model's ability to generalize to new, unseen data.
* **Purpose**: To assess how well the model performs on data it has not seen during training. This helps in understanding the model's generalization capability and identifying potential overfitting.

**3. Avoiding Overfitting:**

* **Overfitting**: When a model learns the training data too well, including the noise and outliers, it may perform very well on the training data but poorly on new, unseen data. Overfitting means the model has low bias but high variance.
* **Testing Data**: Helps to detect overfitting. If the model performs well on the training data but poorly on the testing data, it indicates that the model is overfitting.

**4. Hyperparameter Tuning:**

* **Validation Set**: Sometimes, the data is further split into a validation set (in addition to training and testing sets). The validation set is used to tune hyperparameters (model parameters that are not learned during training but set before the training process).
* **Purpose**: To fine-tune the model and choose the best hyperparameters without using the testing data, preserving the integrity of the final evaluation.

**5. Measuring Performance Metrics:**

* **Accuracy, Precision, Recall, F1-Score, etc.**: These metrics are calculated on the testing data to provide an unbiased estimate of the model's performance.
* **Purpose**: To ensure that the model's reported performance metrics are realistic and not overly optimistic.

**Why Implement Gradient Descent in the fit() Function**

The fit() function is where the model learns from the training data. Implementing gradient descent in the fit()function is necessary for the following reasons:

1. **Parameter Optimization**:
   * **Objective**: To find the parameters (weights) that minimize the cost function (usually the Mean Squared Error in linear regression).
   * **Cost Function**: J(θ)=12m∑i=1m(hθ(x(i))−y(i))2J(θ)=2m1​∑i=1m​(hθ​(x(i))−y(i))2
     + hθ(x)hθ​(x) is the hypothesis (prediction).
     + θθ are the parameters.
     + mm is the number of training examples.
2. **Iterative Improvement**:
   * Gradient descent is an iterative algorithm that starts with an initial guess for the parameters and repeatedly adjusts them to reduce the cost function.
   * In each iteration, the parameters are updated using the gradient of the cost function with respect to the parameters.
   * Update rule: θj:=θj−α∂∂θjJ(θ)θj​:=θj​−α∂θj​∂​J(θ)
     + αα is the learning rate.
     + ∂∂θjJ(θ)∂θj​∂​J(θ) is the partial derivative of the cost function with respect to the parameter θjθj​.
3. **Convergence to Optimal Solution**:
   * The iterative process continues until convergence, meaning the cost function does not decrease significantly with further iterations, indicating the parameters have reached optimal values.

**Implementation Outline**

Here is a conceptual outline of implementing gradient descent within the fit() function of a linear regression model:

1. **Initialize Parameters**:
   * Start with initial guesses for the parameters, typically zeros or small random values.
2. **Compute the Cost Function**:
   * Calculate the cost function using the current parameters to assess how well the model is performing.
3. **Update Parameters**:
   * Compute the gradients (partial derivatives) of the cost function with respect to each parameter.
   * Adjust the parameters in the direction that reduces the cost function.
4. **Repeat**:
5. Continue the iterative process until convergence.
6. **Initialization**: Parameters are initialized to zeros.
7. **Cost Function**: Calculated to track the performance of the model during training.
8. **Gradient Calculation**: Partial derivatives of the cost function with respect to each parameter are computed.
9. **Parameter Update**: Parameters are updated in the direction that decreases the cost function.
10. **Convergence**: The process repeats until the parameters converge to their optimal values.

By implementing gradient descent in the fit() function, the linear regression model can effectively learn from the training data and find the optimal parameters that minimize prediction error.