Logistic Regression

Logistic Regression is a supervised learning algorithm used primarily for **classification problems**. Despite its name, logistic regression is not used for regression tasks but for binary and multiclass classification. It estimates the probability that a given input belongs to a particular class, using a logistic (sigmoid) function to map predicted values to a range between 0 and 1.

The basic idea behind logistic regression is to model the relationship between one or more independent variables (features) and a dependent variable (target), where the target is categorical.

The formula for logistic regression is:

$p = 1 / (1 + e\text{^}-(b_0 + b_1x_1 + b_2x_2 + ... + b_nx_n))$

Where:

- (p) is the predicted probability,
- (b_0) is the intercept (bias),
- (b_1, b_2, ..., b_n) are the coefficients (weights) for the input features (x_1, x_2, ..., x_n),
- (e) is the base of the natural logarithm (Euler’s number).

Classification Problem

A **classification problem** refers to a task where the goal is to assign a label from a predefined set of categories to input data. The output in classification is discrete (e.g., yes/no, spam/not spam, class labels).

Types of Classification Problems:

1. **Binary Classification**:

   - This is the simplest classification task, where the target variable has only **two possible classes**. For example, predicting whether an email is spam or not (spam/non-spam).
   - Logistic regression is frequently used for binary classification.

   Example: Predicting if a patient has a disease (yes/no).

2. **Multiclass Classification**:

   - This involves **more than two classes**. The goal is to assign one class label from multiple possible categories.
   - Logistic regression can be extended to handle multiclass classification using strategies like **one-vs-rest (OvR)** or **softmax regression**.

   Example: Classifying types of fruit (apple/orange/banana).

Sigmoid Function

The **sigmoid function** is a mathematical function used in logistic regression to map predicted values to probabilities. It has an "S" shaped curve and squashes input values to the range (0, 1). This is essential for binary classification tasks where you want to predict the probability of class membership.

The formula for the sigmoid function is:

σ(z) = 1 / (1 + e^-z)

Where:

- (z) is the linear combination of input features and weights (i.e., the output from the linear part of logistic regression).
- (\sigma(z)) is the probability value between 0 and 1.

The sigmoid function helps in decision-making by setting a threshold (usually 0.5). If (\sigma(z) > 0.5), the model predicts one class (e.g., positive), otherwise it predicts the other class (e.g., negative).

Summary

- **Logistic Regression**: A classification algorithm that predicts probabilities using the sigmoid function.
- **Classification Problem**: A task of assigning input data to a set of predefined categories.
- **Binary Classification**: Two possible output classes.
- **Multiclass Classification**: More than two possible output classes.
- **Sigmoid Function**: Maps any real-valued number into a probability between 0 and 1.

These concepts form the basis of many classification tasks in machine learning.

Cost Function of Logistic Regression

The **cost function** in logistic regression, also known as **log loss** or **binary cross-entropy**, measures how well the model's predicted probabilities match the actual labels. The goal of the cost function is to penalize wrong predictions by computing the difference between the predicted output and the actual output.

The logistic regression cost function is derived from the **likelihood** of the data given the parameters. It helps the model adjust its parameters (weights) during training to improve the predictions.

Formula for Cost Function:

The cost function for a single training example in logistic regression is:

```
Cost(hθ(x), y) = −[y ∗ log(hθ(x)) + (1 − y) ∗ log(1 −
hθ(x))]
```

- **hθ(x)**: The predicted probability (output of the sigmoid function).
- **y**: The actual label (either 0 or 1).

The overall cost function (also known as the **loss function**) for the entire dataset is the average of the individual costs over all training examples. This can be written as:

```
J(θ) = −1/m ∗ Σ [y(i) ∗ log(hθ(x(i))) + (1 − y(i)) ∗ log(1
− hθ(x(i)))]
```

Where:

- **J(θ)**: The cost function.
- **m**: Number of training examples.
- **Σ**: Summation over all training examples.
- **hθ(x(i))**: The predicted probability for the i-th training example.
- **y(i)**: The actual label for the i-th training example.

Explanation of the Components:

- **log(hθ(x))**: This term is used when the actual label `y = 1`. It calculates the log of the predicted probability. If the prediction is correct and close to 1, this term will be close to 0 (no penalty). If the prediction is wrong (close to 0), the cost increases dramatically.

- **log(1 - hθ(x))**: This term is used when the actual label `y = 0`. It calculates the log of `1 − predicted probability`. If the prediction is close to 0 (correct prediction), the cost will be low. If the prediction is close to 1 (wrong prediction), the cost will be high.

Intuition Behind the Cost Function:

- When the predicted probability (hθ(x)) is close to the actual label (y), the cost is small.
- When the predicted probability is far from the actual label, the cost increases rapidly.

The cost function is designed to handle the non-linearity of the logistic regression model, which is essential because the output is a probability between 0 and 1.

Graph of the Cost Function:

- The cost function is **convex**, meaning it has a single global minimum, which is the point the optimization algorithm (like Gradient Descent) tries to reach. This ensures that the model can find the best possible parameters without getting stuck in local minima.

Summary:

- The cost function in logistic regression is based on the **log loss** or **binary cross-entropy**.
- It penalizes wrong predictions more severely the further they are from the actual labels.
- The optimization algorithm minimizes the cost function to improve the accuracy of the model's predictions.

This function plays a crucial role in guiding the model during training to find the best-fit line that maximizes prediction accuracy.

Mean Squared Error (MSE) is not suitable for Logistic Regression

The **Mean Squared Error (MSE)** is not suitable for logistic regression because of several key reasons related to the nature of the task, the type of output, and how the optimization process works in logistic regression. Here's why:

1. **Non-Linearity of Sigmoid Function**

- Logistic regression uses a **sigmoid function** to output probabilities between 0 and 1. The sigmoid function is **non-linear**, which means that it transforms the input (linear combination of features) into a non-linear space.
- The MSE is designed for linear models, where the relationship between input and output is assumed to be linear. Using MSE with the non-linear sigmoid function would complicate the gradient descent optimization, leading to slower convergence and poor performance.

2. **Non-Convexity of MSE in Logistic Regression**

- For linear regression, the MSE cost function is **convex**, meaning it has a single global minimum, which makes it easy for optimization algorithms like Gradient Descent to find the best parameters.
- However, when applied to logistic regression, the MSE cost function becomes **non-convex** due to the non-linear nature of the sigmoid function. Non-convex functions have multiple local minima, which can cause the optimization algorithm to get stuck in a local minimum, preventing it from finding the optimal solution.

3. **Asymmetry of the Error**

- Logistic regression outputs **probabilities**, and we are interested in measuring how well the model's predicted probability fits the true label (which is 0 or 1).
- MSE treats the difference between predicted and actual values symmetrically, meaning the penalty for a prediction of 0.9 when the actual value is 1 is the same as the penalty for a prediction of 0.1 when the actual value is 0. However, in

classification tasks, this symmetric behavior is not ideal because the nature of classification errors is different from regression errors.

4. **Interpretation of Probabilities**

- Logistic regression deals with probabilities, not continuous values as in regression. Using MSE would imply that the output can take any continuous value, which is not appropriate for a probability model. MSE tries to minimize the difference between continuous values, but in logistic regression, we want to minimize the difference between **probabilities** and the actual binary outcomes (0 or 1).

5. **Logarithmic Loss is More Appropriate**

- Instead of MSE, **logarithmic loss** (also known as log loss or binary cross-entropy) is used in logistic regression because it is designed to handle probabilities. Log loss provides a **convex** function, making optimization straightforward with a global minimum, and it asymmetrically penalizes wrong predictions more appropriately based on the nature of classification.

Summary

- **MSE** is designed for **continuous** outputs in **regression** tasks and assumes a linear relationship between inputs and outputs.
- Logistic regression is a **classification** algorithm that outputs **probabilities** in a **non-linear** space.
- MSE would lead to **non-convexity**, slow or problematic convergence during optimization, and an inappropriate error interpretation for classification tasks.
- **Log Loss** (binary cross-entropy) is more suitable for logistic regression because it is convex, handles probabilities correctly, and better fits the classification nature of the problem.

This is why **MSE is not suitable for logistic regression**. Instead, log loss is the proper cost function to optimize.

Confusion Matrix in Model Evaluation

The **confusion matrix** is a tool used to evaluate the performance of a **classification model**. It provides a detailed breakdown of the predictions made by the model, comparing the predicted class labels to the actual class labels. This helps assess not only the accuracy of the model but also how well it distinguishes between different classes.

Structure of a Confusion Matrix

For **binary classification**, the confusion matrix is typically a 2x2 table where:

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

- **True Positive (TP)**: The model correctly predicted the positive class.
- **True Negative (TN)**: The model correctly predicted the negative class.
- **False Positive (FP)**: The model incorrectly predicted the positive class (Type I error).
- **False Negative (FN)**: The model incorrectly predicted the negative class (Type II error).

Metrics Derived from the Confusion Matrix

The confusion matrix can be used to calculate several key metrics for evaluating the classification model's performance:

1. **Accuracy**: The proportion of correctly predicted instances (both true positives and true negatives).

   `Accuracy = (TP + TN) / (TP + TN + FP + FN)`
2. **Precision**: The proportion of true positive predictions among all positive predictions (focuses on how many of the predicted positives are actually positive).

   `Precision = TP / (TP + FP)`
3. **Recall (Sensitivity or True Positive Rate)**: The proportion of actual positive cases that were correctly predicted (focuses on how many actual positives were identified).

   `Recall = TP / (TP + FN)`
4. **F1-Score**: The harmonic mean of precision and recall, used when you want to balance both metrics.

   `F1-Score = 2 * (Precision * Recall) / (Precision + Recall)`
5. **Specificity (True Negative Rate)**: The proportion of actual negative cases that were correctly predicted.

   `Specificity = TN / (TN + FP)`
6. **False Positive Rate (FPR)**: The proportion of negative cases that were incorrectly predicted as positive.

   `FPR = FP / (FP + TN)`

Why Confusion Matrix is Used Only for Classification Models

1. **Discrete Output**:

   - Classification models predict **discrete classes** or labels, such as "spam" vs. "not spam" or "cat" vs. "dog". The confusion matrix compares these predicted labels against the true labels.

- In contrast, **regression models** predict **continuous values**, such as predicting house prices or temperatures, and there's no clear distinction between discrete classes in such predictions. For continuous outputs, metrics like **Mean Squared Error (MSE)** or **Root Mean Squared Error (RMSE)** are used instead.

2. **Evaluating Classifications**:
   - The confusion matrix provides detailed insight into how well the model differentiates between classes, including false positives and false negatives, which are specific to classification tasks.
   - It helps identify specific patterns of misclassification, such as whether the model is systematically confusing certain classes, and this information can be used to improve the model.
3. **Binary and Multiclass Suitability**:
   - The confusion matrix can be extended beyond binary classification to **multiclass classification**. For instance, in a 3-class problem, the confusion matrix would be a 3x3 table, with each row and column representing one of the classes.

Summary

The **confusion matrix** is a powerful tool for evaluating the performance of classification models because it provides a clear breakdown of **true positives**, **true negatives**, **false positives**, and **false negatives**. These components allow for the calculation of important evaluation metrics such as **precision**, **recall**, **accuracy**, and **F1-score**. It is specifically used for classification models because it deals with **discrete class labels**, which are not present in regression tasks where continuous outputs are involved.