UNIVERSIDAD POLITÉCNICA DE YUCATAN

ROBOTICS COMPUTATIONAL ENGINEERING

MACHINE LEARNING

PROFESSOR: VICTOR ORTIZ

STUDENTS: JESUS GABRIEL CANUL CAAMAL

GRADE: 9° GROUP: B

SCHOOLAR CYCLE: 2023-2024 B

PERCEPTRON IMPLEMENTATION

The Perceptron algorithm stands as a cornerstone in the realm of supervised learning, particularly when it comes to binary classification tasks. It operates on the intriguing premise that if it's feasible to draw a clear boundary, in the form of a hyperplane, between two distinct classes within the feature space, then a Perceptron model can be engineered to learn this boundary effectively. This simple yet powerful concept underpins the Perceptron's decision-making process.

Delving Deeper into the Perceptron's Intuition:

The core intuition behind the Perceptron is rooted in the possibility of finding a linear separation between classes. It hinges on the belief that a hyperplane can be positioned in the feature space to act as a decisive boundary for classifying data points. This hyperplane, essentially a higher-dimensional plane, is represented by a linear combination of input features. The Perceptron's primary objective is to fine-tune the weights associated with these features in a way that enables it to make accurate classifications.

During the training phase, the Perceptron engages in a dynamic learning process. It continually adapts its internal parameters, particularly the weights, by iterating over the training data. This iterative process allows the Perceptron to minimize classification errors gradually. The algorithm is not concerned with finding the absolute optimal solution but rather seeks to establish a hyperplane that categorizes data points into their respective classes with minimal error.

PSEUDOCODE

```
W= represents the weight vector.
B= the bias term.
Alpha= learning rate.
(x, y)= represents a training example.
Z= weighted sum of inputs.

Initialize weights (w) and bias (b) to small random values or zeros
Set learning rate (alpha)
Repeat until convergence:
    for each training example (x, y):
        Compute the weighted sum of inputs:
```

```
    z = w * x + b
Apply the activation function (e.g., step function):
    if z > 0:
        predicted_class = 1
    else:
        predicted_class = 0
Update the weights and bias:
    w = w + alpha * (y - predicted_class) * x
    b = b + alpha * (y - predicted_class)
```

LOSS FUNCTION + OPTIMIZATION FUNCTION IDENTIFICATION OF KNN

Loss Function and Optimization Function in Machine Learning are critical components of the training process, each with its own distinct role. Let's delve deeper into these concepts:

Loss Function:

The Loss Function, also known as a cost or objective function, serves as a quantitative measure of the error or mismatch between the model's predictions and the actual ground truth. In the context of the Perceptron, it plays a fundamental role in guiding the model towards making better predictions.

In the provided Perceptron example, the loss function is simplistic, summing up the discrepancies between the true labels (y_true) and the predicted labels (y_pred) for each data point. In this particular case:

Loss = Σ(y_true - y_pred)

This simple loss function counts how many times the Perceptron misclassifies the training data. The aim of the Perceptron's training process is to minimize this loss, which means reducing the number of misclassified examples.

In more complex machine learning models and tasks, the loss function can take various forms tailored to the specific problem. For instance, in regression tasks, Mean Squared Error (MSE)

is often used, while cross-entropy loss is common in classification tasks. The choice of the loss function is driven by the problem's nature and the desired model behavior.

Optimization Function:

Unlike deep learning models that employ optimization algorithms like Stochastic Gradient Descent (SGD), the Perceptron doesn't employ traditional optimization functions in the same way. Instead, it uses a simple weight update rule that directly adjusts the model's parameters (weights and bias).

The optimization function in deep learning typically involves iteratively adjusting model parameters to minimize the loss function. Techniques like SGD, Adam, RMSprop, and more are used to guide this parameter optimization process, making incremental updates based on gradients.

In contrast, the Perceptron updates its weights and bias directly through its learning rule. When a misclassification occurs, the weights and bias are adjusted in proportion to the error, which is determined by the difference between the true label and the predicted label. This direct update strategy is a fundamental characteristic of the Perceptron's learning process.

However, it's essential to note that the Perceptron is a linear model, capable of handling only linearly separable problems. For more intricate and non-linear tasks, more advanced models like Multi-Layer Perceptrons (MLPs), accompanied by sophisticated optimization functions, are employed to navigate the complex decision boundaries inherent in those scenarios.

In summary, the Loss Function quantifies the error in model predictions, while the Optimization Function, in the context of the Perceptron, is replaced by a simple weight update rule. For complex, non-linear tasks, more advanced models and optimization algorithms are needed to achieve accurate and robust results.