Demystifying the Perceptron Algorithm

Rishabh Iyer

March 19, 2024

1 Introduction

The Perceptron is a fundamental building block in the field of machine learning, representing one of the earliest forms of artificial neural networks. Conceived by Frank Rosenblatt in 1957, the Perceptron is a type of linear classifier that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The algorithm enables the learning of these weights by iteratively adjusting them in response to the error of the output compared to the expected result. Despite its simplicity and limitations in solving only linearly separable problems, the Perceptron laid the groundwork for the development of more complex neural networks and remains a pivotal concept in understanding the basics of machine learning algorithms.

In this blog article, we will focus on *Binary Classification*. For binary classification, we have two classes: positives and negatives. In other words, $y \in \{-1, \}$ which means y can be either -1 or $+1$.

2 A Deep Dive Into Perceptrons

TODO: Reference blog article 2 that discusses the five step recipie for creating ML models.

Let's explore the components of Perceptron in detail, using spam classification as an illustrative example.

2.1 Step 1: Data Preparation and Gathering

In spam classification, the goal is to categorize emails into "spam" or "not spam" (ham). The data preparation involves selecting features that are indicative of spam content and labeling the data accordingly.

Features: Common features include the frequency of certain keywords ("free", "offer", "winner", "viagra"), the sender's reputation, the presence of attachments, and the use of capital letters. These features are encoded numerically to serve as input (x) to the model.

Labels: Each email is labeled as $+1$ (spam) or -1 (not spam), serving as the target variable (y) for the classification task.

Next, we collect a training dataset comprising a large number of x, y pairs:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \cdots, (x^{(M)}, y^{(M)})\}$$

M is the total number of training examples in the dataset. Taking the example of spam classification, we have M emails in our training dataset of comprising of both spam emails and not spam emails with their labels.

2.2 The Linear Model

The Perceptron model attempts separate between the positive and negative class. The model is defined as:

$$f(x) = \text{sign}(w^T x + b)$$

where w is the weight vector, b is the bias, and $\text{sign}(\cdot)$ is an activation function that maps the weighted sum of the inputs to $+1$ or -1, i.e., positives and negatives.

Combing back to the spam email classification application, the model attempts to separate the spam and ham emails using a linear decision boundary $f(x) = \text{sign}(w^T x + b)$. Once the model which comprises of w, b is learnt (using the Loss function and optimization described below), $f(x)$ will then be the predicted label (+1 means spam and -1 means ham).

2.3 The Loss Function

The initial intuition might be to use the 0/1 loss, which penalizes misclassifications:

$$L_{0/1}(w, b) = \frac{1}{M} \sum_{i=1}^{M} \mathbb{I}(y^{(i)} \neq f(x^{(i)}))$$

where $\mathbb{I}(\cdot)$ is the indicator function. However, the 0/1 loss is non-continuous and non-differentiable, making it unsuitable for optimization via gradient-based methods.

To overcome this, the Perceptron loss is introduced:

$$L_{\text{Perceptron}}(w, b) = \sum_{i=1}^{M} \max(0, -y^{(i)}(w^T x^{(i)} + b))$$

Note the term: $\max(0, -y^{(i)}(w^T x^{(i)} + b))$. When $y^{(i)}(w^T x^{(i)} + b) \geq 0$, both y^i and $w^T x^{(i)} + b$ are of the same sign, which means it is **correctly classified**. In this case, the loss is zero. When $y^{(i)}(w^T x^{(i)} + b) < 0$, it means that y^i and $w^T x^{(i)} + b$ are of different signs, which implies this example is **misclassified**. So essentially, this loss function only sums over misclassified examples, and for every misclassified example, the loss is $-y^{(i)}(w^T x^{(i)} + b)$, which is a positive number.

So as you can imagine, the loss function is encouraging the model to adjust w and b to correctly classify those examples.

Figure 1 compares the Perceptron loss with the 0/1 loss. The 0/1 Loss penalizes all misclassifications equally, while the Perceptron Loss penalizes misclassifications in proportion to their distance from the decision boundary.

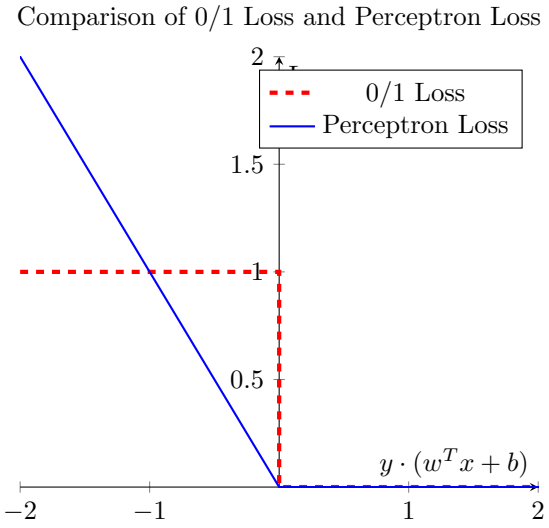

Figure 1: Illustration of the 0/1 Loss and Perceptron Loss as functions of the product $y \cdot (w^T x + b)$. The 0/1 Loss penalizes all misclassifications equally, while the Perceptron Loss penalizes misclassifications in proportion to their distance from the decision boundary.

2.4 The Optimization Algorithm

The Perceptron algorithm (which is the optimization algorithm that minimizes the perceptron loss described above) is a foundational method in machine learning for binary classification. It updates the model's weights and bias to minimize a specific loss function known as the Perceptron loss. The development of this algorithm can be understood by examining the subgradient of the Perceptron loss and recognizing the algorithm as an application of stochastic subgradient descent.

The Perceptron loss for a single training example $(x^{(i)}, y^{(i)})$ is defined as:

$$L_{\text{Perceptron}}(w, b) = \max(0, -y^{(i)}(w^T x^{(i)} + b))$$

This loss is zero for correctly classified examples where $y^{(i)}(w^T x^{(i)} + b) > 0$, and linearly increases with the magnitude of $y^{(i)}(w^T x^{(i)} + b)$ for misclassified examples.

2.4.1 Subgradient of the Perceptron Loss

To perform gradient-based optimization, we need to compute the gradient (or subgradient, to be precise, due to the non-differentiability at zero) of the Perceptron loss with respect to w and b. For a misclassified example where $y^{(i)}(w^T x^{(i)} + b) < 0$, the subgradient of the Perceptron loss with respect to w is:

$$\nabla_w L_{\text{Perceptron}} = -y^{(i)} x^{(i)}$$

And with respect to b:

$$\nabla_b L_{\text{Perceptron}} = -y^{(i)}$$

For correctly classified examples, where the loss is zero, the subgradient with respect to both w and b is zero.

To understand the gradient of this loss with respect to the weights w and bias b, we need to consider the piecewise nature of the max function. The derivative of $\max(0, -y(w^T x + b))$ with respect to w depends on which argument of the max function is larger.

If $-y(w^T x + b) < 0$ (**correctly classified case**), the max function outputs 0, and the function is effectively constant with respect to w. Thus, its derivative is 0.

If $-y(w^T x + b) > 0$ (**mis-classified case**), the max function outputs $-y(w^T x + b)$, and we need to compute the derivative of this expression with respect to w. Using the chain rule, we get:

$$\frac{\partial}{\partial w}[-y(w^T x + b)] = -yx$$

Similarly, the derivative with respect to b when $-y(w^T x + b) > 0$ is:

$$\frac{\partial}{\partial b}[-y(w^T x + b)] = -y$$

Therefore, the subgradient of the Perceptron loss with respect to w and b for a misclassified example (where $-y(w^T x + b) > 0$) is $-yx$ and $-y$, respectively. This reflects the direction in which w and b should be adjusted to increase the margin for that misclassified example, pushing it towards the correct side of the decision boundary. For the case when $y(w^T x + b) = 0$, we can set the subgradient to be 0.

2.4.2 Perceptron as Stochastic Subgradient Descent

TODO: Reference the gradient descent blog article which goes over stochastic gradient descent.

The Perceptron algorithm updates the weights and bias by considering one training example at a time, making it a form of stochastic subgradient descent. Specifically, the update rule:

$$w := w + \eta y^{(i)} x^{(i)}$$

$$b := b + \eta y^{(i)}$$

can be seen as applying a step in the direction opposite to the subgradient of the Perceptron loss for misclassified examples, with η acting as the learning rate. This update rule directly minimizes the Perceptron loss by adjusting w and b to correct the misclassification of the current example.

The perceptron algorithm is described in the following steps:

1. Initialize w and b to zeros or small random values.
2. For each training example, update w and b if it is misclassified:

$$w := w + \eta y^{(i)} x^{(i)}$$

$$b := b + \eta y^{(i)}$$

   where η is the learning rate.

3. Repeat until the model converges (all points correctly classified) or a maximum number of iterations is reached.

2.4.3 Standard Subgradient Descent

While the Perceptron algorithm traditionally updates its parameters based on individual examples (stochastic subgradient descent), it is also possible to apply standard (batch) subgradient descent to the Perceptron loss. In this approach, the subgradient of the loss is computed over the entire training set, and the parameters are updated based on the average subgradient. This method can offer more stable updates at the cost of increased computational effort per iteration, as it requires processing the entire dataset to compute each update.

The Perceptron algorithm's development from the Perceptron loss and its interpretation as a stochastic subgradient descent method highlight its theoretical foundation and practical effectiveness in linear classification tasks. By iteratively minimizing the Perceptron loss, the algorithm seeks to find a linear boundary that separates the classes in the training data.

2.4.4 Convergence of the Perceptron Algorithm

A remarkable property of the Perceptron algorithm is its convergence guarantee under certain conditions. Specifically, if the dataset is linearly separable, the Perceptron algorithm is guaranteed to find a hyperplane that perfectly separates the two classes, achieving zero training error.

A dataset is said to be *linearly separable* if there exists at least one hyperplane that can separate the data points of one class from those of the other class without error. The Perceptron algorithm leverages this property by iteratively adjusting the weights and bias to minimize classification errors.

Theorem: For a linearly separable dataset, the Perceptron algorithm converges to a weight vector w and bias b that correctly classify all training examples, after a finite number of iterations.

Proof Sketch: The convergence proof relies on the notion that, with each update, the Perceptron algorithm makes progress towards the optimal separator. Specifically, it can be shown that the algorithm reduces the misclassification margin (the distance of misclassified points to the decision boundary) with each update. Since the dataset is finite and linearly separable, there is a maximum number of updates after which no further improvements can be made, leading to convergence.

The convergence property of the Perceptron algorithm has several important implications:

- **Guarantee of Solution:** For linearly separable datasets, the Perceptron algorithm is guaranteed to find a separating hyperplane, ensuring that a solution exists and can be found.
- **Finite Iterations:** The algorithm will converge in a finite number of steps, although the exact number of iterations required may depend on the initial weights, the learning rate, and the geometry of the dataset.
- **Sensitivity to Data and Initialization:** While convergence is guaranteed, the final model may vary based on the initial conditions and the order in which examples are presented. This can affect the margin of the resulting separator and, potentially, its generalization to new data.

2.5 Evaluation Criteria and Inference

At test time, given a test example x_{test}, we can obtain the predicted label y_{pred} as follows:

$$y_{\text{pred}} = w^T x_{\text{test}} + b \tag{1}$$

where (w, b) are the weights and bias terms obtained from the optimization (step 4) of the loss function (step 3) outlined earlier – a combination of step 3 and step 4 is known as *training*. Once, we obtain the predicted labels on a held-out test dataset, we can evaluate the performance of the model. The Performance of the model can be evaluated using metrics such as accuracy, precision, recall, and the F1 score. For spam classification, precision (the proportion of true spam emails among those labeled as spam) and recall (the proportion of true spam emails that were correctly identified) are particularly important to balance the cost of misclassifying ham as spam and vice versa. The metrics below can be applied to any binary classification task and will hold for Support Vector Machines and other machine learning models discussed in future blog articles.

Accuracy:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Precision and Recall:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1 Score:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

These metrics provide a comprehensive view of the model's performance, highlighting its strengths and areas for improvement in the context of spam classification.

3 Summary of the Perceptron Algorithm

The Perceptron algorithm is a fundamental approach to binary classification in machine learning. Here is a concise summary of the steps involved in building and evaluating a Perceptron model:

1. **Collect the Dataset:** Obtain a dataset consisting of input feature vectors (x) and corresponding binary target labels (y), where $y \in \{-1, 1\}$ for a two-class classification problem.

2. **Initialize the Model:** Start with initial values for the weight vector (w) and bias (b). These can be set to zeros or small random numbers.

3. **Select the Loss Function:** The Perceptron algorithm typically uses the Perceptron loss, which is designed to penalize misclassified examples. The loss for an individual example is given by:

$$L_{\text{Perceptron}}(w, b) = \max(0, -y^{(i)}(w^T x^{(i)} + b))$$

4. **Optimize Using (Stochastic) Gradient Descent (SGD):** The Perceptron algorithm updates the weights and bias for each misclassified example in the training set, using the rule:

$$w := w + \eta y^{(i)} x^{(i)}$$

$$b := b + \eta y^{(i)}$$

where η is the learning rate, typically set to 1 in the basic Perceptron algorithm. This process is repeated for a fixed number of iterations or until the dataset is perfectly classified. Alternatively, a full subgradient descent algorithm can also be employed although the traditional perceptron algorithm is the stochastic version.

5. **Training Procedure Output:** The result of the training procedure (the above two steps, i.e., optimization on the selected loss function) is a linear classifier defined by the optimized parameters (w, b), which can separate the two classes with a decision boundary.

6. **Predict on Test Dataset:** Use the trained Perceptron model to classify new, unseen examples in the test dataset. The prediction for a test input x is given by the sign of $w^T x + b$.

7. **Evaluate the Model:** Assess the performance of the Perceptron model on the test dataset using metrics such as accuracy, precision, recall, and the F1 score. These metrics help determine the effectiveness of the model in classifying new examples.

This procedure outlines the steps to implement and evaluate a Perceptron model, from data preparation through to model assessment. It highlights the simplicity and efficiency of the Perceptron in solving linearly separable binary classification problems.

4 Pros and Cons of the Perceptron Algorithm

The Perceptron algorithm, as one of the earliest algorithms in machine learning, has several notable advantages and disadvantages. Understanding these can help in determining when and how to apply this algorithm effectively.

4.1 Advantages

- **Simplicity:** The Perceptron algorithm is straightforward to implement and understand, making it an excellent introduction to the concepts of machine learning and linear classification.

- **Efficiency:** Due to its simplicity, the Perceptron is computationally efficient, requiring relatively few resources to run, which makes it suitable for datasets of moderate size and for real-time processing.

- **Convergence Guarantee:** For linearly separable datasets, the Perceptron algorithm is guaranteed to converge to a solution that perfectly separates the classes, within a finite number of iterations.

- **Online Learning:** The Perceptron can be easily adapted for online learning, allowing it to update the model incrementally as new data arrives, which is valuable in environments where data is received sequentially.

4.2 Disadvantages

- **Limited to Linearly Separable Problems:** The Perceptron algorithm can only guarantee convergence for linearly separable datasets. It struggles with non-linearly separable data, often failing to find a satisfactory solution. In cases where the data is not linearly separable, the Perceptron can continue updating indefinitely without converging to a stable solution, necessitating the use of additional stopping criteria.

- **Arbitrary Decision Boundary:** One of the most significant limitations of the Perceptron algorithm is its inability to choose among multiple perfect classifiers. When the dataset is linearly separable, there may be many hyperplanes that can perfectly separate the classes. The Perceptron algorithm does not have a mechanism to select the optimal or most robust classifier among these. As a result, it often ends up selecting a decision boundary that is too close to one of the classes, which can adversely affect the model's generalization ability on unseen data. Support Vector Machines (SVMs) and other max-margin classifiers address this key issue by not just seeking any separating hyperplane, but the one that maximizes the margin between the classes. This approach tends to result in better generalization on test data, as the decision boundary is as far away as possible from any data point, reducing the risk of misclassification.

- **Sensitivity to Outliers:** The final model can be significantly influenced by outliers, as the Perceptron seeks to classify every example correctly, including outliers that might be better treated as exceptions.

- **No Probability Estimates:** Unlike some other classifiers (like logistic regression and naive bayes), the Perceptron does not provide probabilities for class memberships, only hard classifications. This can be a limitation in applications where understanding the confidence of predictions is important.