CHEME 5820: Machine Learning for Engineers **Spring 2025**

Lecture 3a: Linear Regression, Perceptron and Binary Classification

Lecturer: Jeffrey Varner

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications.*

In this lecture, we will discuss the following topics:

- **Supervised learning** is a type of machine learning where the model is trained on labeled data, meaning that each training example includes both the input features (measurements) and the corresponding target (output) variable. The goal is to learn a mapping from inputs to outputs so that the model can make accurate predictions on new, unseen data.
- **Linear regression** is a statistical method for modeling the relationship between a target variable (output) and one or more features (input) by fitting a linear model to observed data. It provides a simple way to predict outcomes and understand relationships between variables. The model is linear in the parameters, not necessarily in the features.
- **Continuous variable prediction tasks**: In machine learning, linear regression models are commonly employed for continuous variable prediction tasks. These models enable the estimation of numerical outcomes based on the (non)linear relationships identified between input features and the target variable.
- **Binary classification tasks**: While linear regression is primarily designed to predict continuous outcomes, it can also be adapted for classification tasks by combining the linear regression model with an output function that transforms the continuous target variable predictions into discrete classes or probability estimates. In this lecture, we'll consider binary classification using the perceptron classification algorithm developed at Cornell in the 1950s.

1 Introduction

This lecture will introduce supervised learning and linear models for regression and classification tasks. The supervised learning problem requires a dataset and a learning algorithm figured reproduced from the Applied Machine Learning (Cornell CS5785, Fall 2024) course:

$$\underbrace{\text{Dataset}}_{\text{Features, Attributes, Targets}} + \underbrace{\text{Learning Algorithm}}_{\text{Model Class + Objective + Optimizer}} \rightarrow \text{Predictive Model}$$

The output of the supervised learning problem is a predictive model that maps inputs to targets. Today, we'll examine linear models. These models are easy to implement and interpret and serve as a foundation for the more complex predictive models we'll explore. In particular, we will discuss estimating the parameters of a linear model for overdetermined systems, and then we'll pivot to the classification problem. However, before getting to the linear models, we will first introduce the optimizer and gradient descent (a quick review of some items from the calculus course you took in high school or college).

2 Optimizer

The material in this section was heavily inspired by the Applied Machine Learning (Cornell CS5785, Fall 2024) course notes. Before we present our first supervised learning algorithm, linear regression, let's do a quick calculus review related to the supervised learning problem.

2.1 Calcular Review

A key part of a supervised learning algorithm is the `optimizer`, which takes an objective J (also called a loss function) and a model $\mathcal{M}$ and tries to find the best model f in $\mathcal{M}$ that minimizes the objective J. In general, an objective function J measures the difference between the predicted values and the observed values in some way, e.g., the mean squared error (MSE), the cross-entropy loss, or the negative log-likelihood. The optimizer outputs a model $f \in \mathcal{M}$ with the smallest value of the objective J, i.e., $\min_{f\in\mathcal{M}} J(f)$. Intuitively, this is the function that best describes the (training) dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, 2, \dots, n\}$. But what is this magical `optimizer` creature? As it turns out, the `optimizer` can be many different things, but in this lecture, we will focus on the `gradient descent optimizer`. This approach uses the gradient of the objective function to find its minimum.

Gradient

The key tool from calculus that we will use to develop the `gradient descent optimizer` is the derivative and its extensions. Suppose we have a univariate function $f_\theta : \mathbb{R} \rightarrow \mathbb{R}$. The derivative $df/d\theta$ (evaluated at θ_0) is the instantaneous rate of change of the function $f(\theta)$ with respect to its parameter θ at the point θ_0. The partial derivative $\partial f(\theta)/\partial\theta_j$ of a multivariate function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is the derivative of f with respect to θ_j while all the other dimensions θ_k for $k \neq j$ are held constant. Finally, the gradient $\nabla_\theta f$ of the multivariate function f with respect to the vector θ is the vector of all the partial derivatives:

$$\nabla_\theta f(\theta) = \begin{bmatrix} \frac{\partial f(\theta)}{\partial\theta_1} \\ \frac{\partial f(\theta)}{\partial\theta_2} \\ \vdots \\ \frac{\partial f(\theta)}{\partial\theta_d} \end{bmatrix} \tag{1}$$

The jth entry of the gradient is the partial derivative $\partial f/\partial\theta_j$ of f with respect to the jth component of θ. We are interested in the gradient because it tells us the direction of the steepest ascent of the function f at the point θ_0. We also know that when the gradient is zero, i.e., $\nabla_\theta f(\theta_0) = \mathbf{0}$, then we have a local minimum or maximum of the function f.

3 Linear Models

Linear models are a class of models used in machine learning for regression tasks, i.e., predicting a continuous output variable from one or more continuous or discrete input features. Linear models assume that the output variable is a linear combination of the input features, i.e., a linear function of the input features weighted by some parameters. Linear in this context is a misnomer because the features are not necessarily linear, but the model is linear in terms of the parameters and features. Suppose we have a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, 2, \dots, n\}$ with n examples, where each example where $\mathbf{x}_i \in \mathbb{R}^m$ is a feature vector and $y_i \in \mathbb{R}$ is the output variable. The linear regression model predicts the output variable y_i for feature vector $\hat{\mathbf{x}}_i$ using the linear function:

$$y_i = \hat{\mathbf{x}}_i^{\top}\beta + \epsilon_i$$

where we have augmented the feature vector with a bias term, i.e., $\hat{\mathbf{x}}_i^{\top} = \left(x_1^{(i)}, \dots, x_m^{(i)}, 1\right)$, $\beta \in \mathbb{R}^p$ is a $p = m + 1$ dimensional column vector of (unknown) parameters to be estimated, and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is a noise term, typically assumed to be a Normal distribution with mean zero and variance σ^2. Depending upon the shape of the data and various other problem constraints, there may be an *analytical solution* for the linear model parameter vector β, e.g., the normal equations, or we can estimate an iterative solution using an optimizer such as gradient descent. Let's first consider the analytical solution for the linear regression parameter vector β for an *overdetermined* system of equations.

3.1 Overdetemined Linear models with regularization

If the number of examples n is greater than the number of features m, the linear model is said to be `overdetermined`. In other words, there are more examples than features. Regularized linear regression models incorporate penalty terms to constrain the size of the coefficient estimates, thereby reducing overfitting and enhancing the model's generalizability to new data. Consider an overdetermined data matrix $\hat{\mathbf{X}} \in \mathbb{R}^{n\times p}$, i.e., the case where $n \gg p$ (there are many more examples than unknown parameters). A regularized least squares estimate of the expected value of the unknown model parameters $\hat{\beta}$ for an overdetermined system will minimize a loss (objective) function of the form:

$$\hat{\beta}_{\lambda} = \arg\min_{\beta} \frac{1}{2} || \mathbf{y} - \hat{\mathbf{X}}\beta ||_2^2 + \frac{\lambda}{2} || \beta ||_2^2 \tag{2}$$

where $|| \star ||_2^2$ is the square of the $L2$-vector norm (Euclidean norm), and $\lambda \geq 0$ denotes a regularization parameter (user-defined hyperparameter), and $\hat{\beta}_{\lambda}$ denotes the estimated expected parameter vector. The expected value of the parameters $\hat{\beta}_{\lambda}$ that minimize the $|| \star ||_2^2$ loss plus penalty for the overdetermined data matrix $\mathbf{X}$ is given by:

$$\hat{\beta}_{\lambda} = \left(\hat{\mathbf{X}}^{\top}\hat{\mathbf{X}} + \lambda \cdot \mathbf{I}\right)^{-1} \hat{\mathbf{X}}^{\top}\mathbf{y}$$

where $\mathbf{I}$ is the identity matrix. The matrix $\hat{\mathbf{X}}^{\top}\hat{\mathbf{X}} + \lambda \cdot \mathbf{I}$ is the `regularized normal matrix`, while $\hat{\mathbf{X}}^{\top}\mathbf{y}$ is the `moment vector`. The inverse $\left(\hat{\mathbf{X}}^{\top}\hat{\mathbf{X}} + \lambda \cdot \mathbf{I}\right)^{-1}$ must exist to obtain the estimated parameter vector $\hat{\beta}_{\lambda}$. If we estimate (or propose) an error model, we can then compute the *uncertainty* of the estimated parameter vector $\hat{\beta}_{\lambda}$ as:

$$\hat{\beta}_{\lambda}^{\dagger} = \hat{\beta}_{\lambda} - \left(\hat{\mathbf{X}}^{\top}\hat{\mathbf{X}} + \lambda \cdot \mathbf{I}\right)^{-1} \hat{\mathbf{X}}^{\top}\epsilon$$

Let's look at the overdetermined linear regression solution without regularization and how the solution is obtained. The steps below can also be applied to the regularized case (but we'll skip the details).

Derivation: Overdetermined solution without regularization

To better understand the overdetermined linear regression solution, let's consider a simple example without regularization, i.e., $\lambda = 0$. The objective function described in Eq. 2 can be rewritten form as:

$$J(\beta) = \frac{1}{2}(\hat{\mathbf{X}}\beta - \mathbf{y})^{\top}(\hat{\mathbf{X}}\beta - \mathbf{y}) \tag{3}$$

We can compute the gradient of the mean squared error as:

$$\begin{aligned}
\nabla_\beta J(\beta) &= \nabla_\beta \frac{1}{2}(X\beta - y)^\top(\hat{\mathbf{X}}\beta - \mathbf{y}) \\
&= \frac{1}{2}\nabla_\beta \left((\hat{\mathbf{X}}\beta)^\top(\hat{\mathbf{X}}\beta) - (\hat{\mathbf{X}}\beta)^\top\mathbf{y} - \mathbf{y}^\top(\hat{\mathbf{X}}\beta) + \mathbf{y}^\top\mathbf{y}\right) \\
&= \frac{1}{2}\nabla_\beta \left(\beta^\top(\hat{\mathbf{X}}^\top\hat{\mathbf{X}})\beta - 2(\hat{\mathbf{X}}\beta)^\top\mathbf{y}\right) \\
&= \frac{1}{2}\left(2(\hat{\mathbf{X}}^\top\hat{\mathbf{X}})\beta - 2X^\top\mathbf{y}\right) \\
&= (\hat{\mathbf{X}}^\top\hat{\mathbf{X}})\beta - \hat{\mathbf{X}}^\top\mathbf{y}
\end{aligned}$$

We used the facts that $a^\top b = b^\top a$ (line 3), that $\nabla_x b^\top \hat{\mathbf{X}} = b$ (line 4), and that $\nabla_x x^\top A x = 2Ax$ for a symmetric matrix A (line 4). Setting the gradient to zero, we get the `normal equations`, which we can solve for the unknown parameters β:

$$\begin{aligned}
(\hat{\mathbf{X}}^\top\hat{\mathbf{X}})\beta - \hat{\mathbf{X}}^\top\mathbf{y} = 0 \\
(\hat{\mathbf{X}}^\top\hat{\mathbf{X}})\beta = \hat{\mathbf{X}}^\top\mathbf{y} \\
\hat{\beta} = (\hat{\mathbf{X}}^\top\hat{\mathbf{X}})^{-1}\hat{\mathbf{X}}^\top\mathbf{y}
\end{aligned}$$

The uncertainty of the estimated parameter vector $\hat{\beta}$ can be computed as (assuming we have an error model with mean zero):

$$\begin{aligned}
\mathbf{y} &= \hat{\mathbf{X}} \cdot \hat{\beta} + \epsilon \\
\hat{\mathbf{X}}^\top\mathbf{y} &= \hat{\mathbf{X}}^\top\hat{\mathbf{X}}\beta + \hat{\mathbf{X}}^\top\epsilon \\
\hat{\mathbf{X}}^\top\hat{\mathbf{X}}\beta &= \hat{\mathbf{X}}^\top\mathbf{y} - \hat{\mathbf{X}}^\top\epsilon \\
\hat{\beta}^\dagger &= \left(\hat{\mathbf{X}}^\top\hat{\mathbf{X}}\right)^{-1}\hat{\mathbf{X}}^\top\mathbf{y} - \left(\hat{\mathbf{X}}^\top\hat{\mathbf{X}}\right)^{-1}\hat{\mathbf{X}}^\top\epsilon
\end{aligned}$$

which is the same as:

$$\hat{\beta}^\dagger = \hat{\beta} - \left(\hat{\mathbf{X}}^\top\hat{\mathbf{X}}\right)^{-1}\hat{\mathbf{X}}^\top\epsilon$$

Error Models

In the previous section, we assumed the error model was a Normal distribution with mean zero and variance σ^2. This is a common assumption in linear regression, but it may not always be true. Therefore, it is a good idea to examine the residuals. The residuals are the difference between the observed values and the predicted values, i.e., $\mathbf{r} = \mathbf{y} - \hat{\mathbf{X}}\hat{\beta}$. If the error model is correct, the residuals should be normally distributed with mean zero and constant variance. We can use a variety of statistical tests to check the normality of the residuals, e.g., the Shapiro-Wilk test, the Kolmogorov-Smirnov test, or the Anderson-Darling test. We can estimate the parameters of the error using maximum likelihood estimation (MLE). We'll talk more about MLE and an alternative approach called maximum a posteriori estimation in a future lecture. So, for now, let's move on to the classification problem.

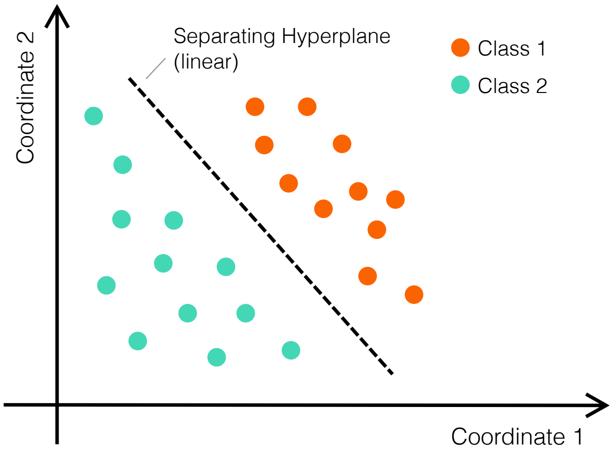


Figure 1: Schematic of a binary classification task with linearly separable data. Above the hyperplane are positive examples, while below the hyperplane are negative examples.

4 The Perceptron and Binary Classification

The Perceptron (1) is a simple yet powerful algorithm used in machine learning for binary classification tasks. The Perceptron (Rosenblatt, 1957) takes the (scalar) output of a linear regression model $y_i \in \mathbb{R}$ and then transforms it using the $\sigma(\star) = \texttt{sign}(\star)$ function to a discrete set of values representing categories, e.g., $\sigma : \mathbb{R} \rightarrow \{-1, 1\}$ in the binary classification case.

Suppose there exists a data set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ with n labeled examples, where each example has been labeled by an expert, i.e., a human to be in a category $\hat{y}_i \in \{-1, 1\}$, given the m-dimensional feature vector $\mathbf{x}_i \in \mathbb{R}^m$. The Perceptron *incrementally learns* a linear decision boundary between two classes of possible objects (binary classification) in $\mathcal{D}$ by repeatedly processing the data. During each pass, a regression parameter vector β is updated until it makes no more than a specified number of mistakes. The Perceptron computes the estimated label $\hat{y}_i$ for feature vector $\hat{\mathbf{x}}_i$ using the $\texttt{sign} : \mathbb{R} \rightarrow \{-1, 1\}$ function:

$$\hat{y}_i = \texttt{sign}\left(\hat{\mathbf{x}}_i^{\top} \cdot \beta\right)$$

where $\beta = (w_1, \dots, w_n, b)$ is a column vector of (unknown) classifier parameters, $w_j \in \mathbb{R}$ corresponding to the importance of feature j and $b \in \mathbb{R}$ is a bias parameter, the features $\hat{\mathbf{x}}_i^{\top} = \left(x_1^{(i)}, \dots, x_m^{(i)}, 1\right)$ are $p = m + 1$-dimensional (row) vectors (features augmented with bias term), and $\texttt{sign}(z)$ is the function:

$$\texttt{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

If data set $\mathcal{D}$ is linearly separable, the Perceptron will incrementally learn a separating hyperplane in a finite number of passes through the $\mathcal{D}$. However, if the data set $\mathcal{D}$ is not linearly separable, the Perceptron may not converge. Psuedo code for the perceptron algorithm is shown in Algorithm 1.

Algorithm 1 The Perceptron Algorithm

```
Input: $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, tolerance $\epsilon \geq 0$, maximum iterations maxiter
Features: $\mathbf{x}_i = (x_{i1}, \ldots, x_{in}, 1)$ are augmented with a bias term, labels $y_i \in \{-1, 1\}$.
Output: Classifier parameters $\beta = (w_1, \ldots, w_n, b)$
$\beta \leftarrow$ rand                          ▷ Initialize parameter vector $\beta$ to a random vector
$i \leftarrow 0$                                 ▷ Initialize the loop counter to zero
while true do                                    ▷ Repeat until stopping criterion is met
    error $\leftarrow 0$                         ▷ Initialize the error count to zero for this pass through $\mathcal{D}$
    for $(\mathbf{x}, y) \in \mathcal{D}$ do     ▷ Iterate over each pair $(\mathbf{x}, y)$ in data set $\mathcal{D}$
        if $y \cdot \left(\mathbf{x}^\top \cdot \beta\right) \leq 0$ then   ▷ Ooops! The data pair $(\mathbf{x}, y)$ is misclassified
            $\beta \leftarrow \beta + y \cdot \mathbf{x}$   ▷ Update the weight vector $\beta$
            error $\leftarrow$ error $+ 1$       ▷ Increment the error count
        end if
    end for
    if error $\leq \epsilon$ or $i \geq$ maxiter then   ▷ Stopping criterion: tolerance or max iterations?
        break                                    ▷ Exit the training loop
    end if
    $i \leftarrow i + 1$                         ▷ Increment the loop counter and repeat
end while
```

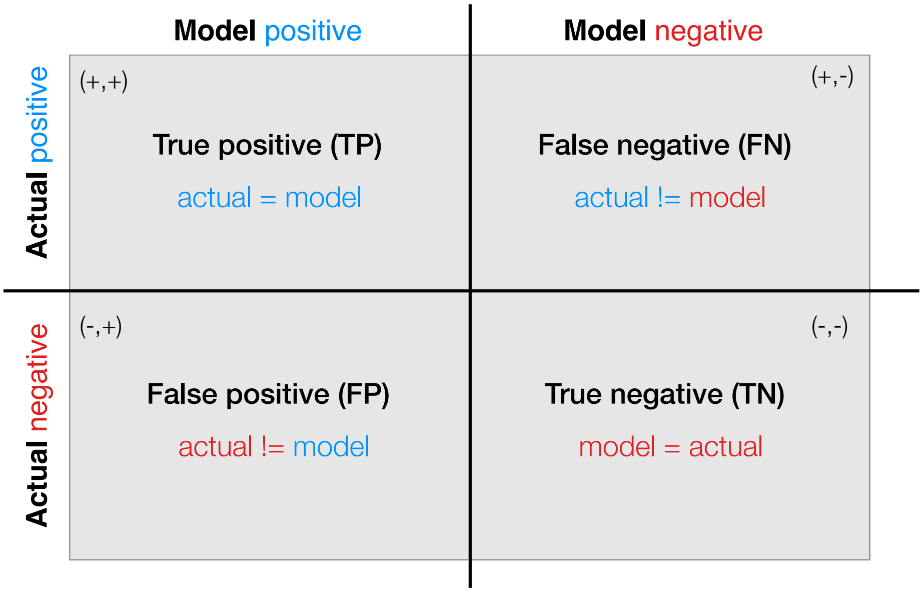
What the algorithm does is to update the weight vector β for each example $(\mathbf{x}_i, y_i)$ in the data set $\mathcal{D}$. If the example is misclassified, i.e., $y_i \left(\mathbf{x}_i^\top \cdot \beta\right) \leq 0$, the weight vector β is updated by adding the feature vector $\mathbf{x}_i$ multiplied by the label y_i. The algorithm continues to iterate through the data set $\mathcal{D}$ until all examples are correctly classified or until a stopping criterion is met, e.g., a maximum number of iterations or a tolerance on the number of classification errors. In the end, the alogrithm estimates a linear hyperplane $\mathcal{H} = \left\{\mathbf{x} \mid \mathbf{x}^\top \cdot \beta = 0\right\}$ that separates the two classes of data points in $\mathcal{D}$ (Fig. 1). The perceptron algorithm is guaranteed to converge to a solution if the data set is linearly separable in a finite number of passes through the data set. For a proof of this convergence guarantee, see the CS4780 lecture notes from 2018.

5 Evaluation of the Binary Classifier

Once the Perceptron (or any binary classifier) has converged, we can evaluate the binary classifier's performance using various metrics. See Sidey-Gibbon et al. (2) for a detailed discussion of these metrics in the context of medical classification problems. The central idea is to compare the predicted labels $\hat{y}_i$ to the actual labels y_i in the data set $\mathcal{D}$. Various metrics can be used to evaluate the performance of a binary classifier, but they all start with computing the confusion matrix.

5.1 Confusion Matrix

The confusion matrix is a table often used to describe the performance of a classification model on a set of data for which the true values are known (Fig. 2). The confusion matrix is a 2×2 matrix that contains four entries: true positive (TP), false positive (FP), true negative (TN), and false negative (FN). The true positive (TP) entry in the confusion matrix is the number of positive examples that were correctly classified as positive. The false negative (FN) entry is the number of positive examples that were incorrectly classified as negative by the model. The false positive (FP) entry is the number of negative examples that were incorrectly classified as positive by the model. The true negative (TN) entry is the number of negative



	Model positive	Model negative
Actual positive	(+,+) True positive (TP) actual = model	(+,-) False negative (FN) actual != model
Actual negative	(-,+) False positive (FP) actual != model	(-,-) True negative (TN) model = actual

Figure 2: Confusion matrix schematic for the binary classification problem.

examples that were correctly classified as negative by the model. The confusion matrix is used to calculate a variety of performance metrics, including accuracy, precision, recall, and the F1 score.

Accuracy. The accuracy of a binary classifier is the proportion of correctly classified *total* examples in the data set. The accuracy is calculated as the sum of the true positive and true negative entries in the confusion matrix divided by the total number of examples in the data set:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

Thus, accuracy measures the classifier's overall performance and indicates how well it can correctly classify examples (both positive and negative).

Precision. The precision of a binary classifier is the proportion of correctly classified *positive* examples out of all examples that were classified as positive by the model. The precision is calculated as the true positive entry in the confusion matrix divided by the sum of the true positive and false positive entries:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5}$$

Precision measures the classifier's ability to classify positive examples correctly, and it is useful when the cost of false positives is high.

Recall (Sensitivity). The recall of a binary classifier is the proportion of correctly classified *positive* examples out of all truly positive examples in the data set. The recall is calculated as the true positive entry in the confusion matrix divided by the sum of the true positive and false negative entries:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6}$$

Recall measures the classifier's ability to correctly classify positive examples, and it is useful when the cost of false negatives is high.

F1 Score. The F1 score of a binary classifier is the harmonic mean of precision and recall. The F1 score is calculated as:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{7}$$

The F1 score measures the classifier's overall performance and balances the trade-off between precision and recall.

6 Summary

This lecture introduced supervised learning and linear models for regression and classification tasks. We discussed overdetermined linear models with regularization and the derivation of the analytical solution for the linear regression parameter vector β. We also introduced a model to compute the uncertainty of the estimated parameter vector $\hat{\beta}$. Finally, we introduced the perceptron algorithm for binary classification problems. The perceptron is a simple linear classifier that can be used to separate two classes of data points. The perceptron algorithm is an iterative algorithm that incrementally updates the weights of the linear classifier to minimize the classification error. Thus, it is one of the first examples of an online learning algorithm, i.e., an algorithm that learns from data incrementally. The perceptron algorithm can converge to a solution if the data set is linearly separable. However, if the data set is not linearly separable, the perceptron algorithm will not converge to a perfect solution (i.e., zero classification error). Suppose we are willing to accept some classification errors. In that case, we can use the perceptron algorithm to find a separating hyperplane in a finite number of passes through the data set, even if the data set is not linearly separable, where some examples are misclassified.

References

1. Rosenblatt F. Perceptron Simulation Experiments. Proceedings of the IRE. 1960;48(3):301–309. doi:10.1109/JRPROC.1960.287598.

2. Sidey-Gibbons JAM, Sidey-Gibbons CJ. Machine learning in medicine: a practical introduction. BMC Med Res Methodol. 2019;19(1):64. doi:10.1186/s12874-019-0681-4.