Lecture 2
Brief Overview of Supervised Machine Learning

By
Dr. Muhammad Alrabeiah
Electrical Engineering Dept., KSU
Spring 2022

January 12th, 2022

Disclaimer These notes are still under development, and they have not been subjected to proper review and revision.

Contents

Chapter 1

Regression

Suggested readings:
Chapter 1 and 3 of textbook [1]

Regression and classification are the most popular tasks in supervised machine learning. This chapter will explore the fundamentals of regression, and more specifically linear regression; it helps build the required background for the more general non-linear regression task, which is the most common in deep learning.

1 Linear Regression

Let's kick off the discussion in this chapter with a definition of what we mean by regression

Definition 1.1. Regression is a supervised machine learning task where a learning algorithm attempts to approximate some *unknown* real-valued function for the purpose of observing some variables and predicting their responses. Formally, let $\mathbf{x} \in \mathbb{R}^{N\times 1}$ be the N-dimensional (N-D) vector of observed variables, let $\mathbf{y} \in \mathbb{R}^{M\times 1}$ be the M-dimensional (M-D) vector of responses, and let $f_t : \mathbf{x} \to \mathbf{y}$ be some unknown real-valued function. Then, the task in regression is to learn a function $f_\Theta : \mathbf{x}, \Theta \to \hat{\mathbf{y}}$ parameterized with a set of parameters $\Theta \in \mathbb{R}^{Q\times 1}$ such that $||\mathbf{y} - \hat{\mathbf{y}}||_d \leq \epsilon$ where ϵ is some arbitrarily small non-negative scaler and $||.||_d$ is some error metric.

Let's try and make sense of the definition above. Consider the example in Figure 1.1.

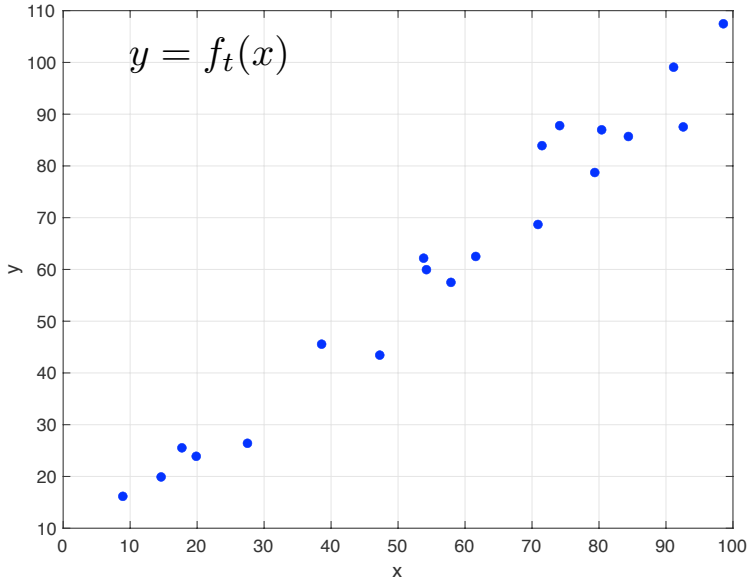


Figure 1.1

The points on the figure are generated by some unknown function $f_t : \mathbf{x} \rightarrow \mathbf{y}$ where $\mathbf{x} = x \in \mathbb{R}, \mathbf{y} = y \in \mathbb{R}$ (both are 1-D). The objective of the learning algorithm is to approximate that unknown function f_t using a dataset of (x, y) data points, i.e., $\mathcal{D} = \{(x, y)_u\}_{u=1}^{U}$. The first step we take is to pick a class of parameterized functions f_Θ for our algorithm to learn. Here we pick

$$\hat{y} = f_\Theta(x) = w_1 x + w_0, \tag{1.1}$$

where $\hat{y} = \hat{\mathbf{y}} \in \mathbb{R}$ and $\Theta = \{w_j\}_{j=0}^{1}$, $w_j \in \mathbb{R}$. The function in Equation 1.1 is linear in the parameters (i.e., Θ), and hence, *the class of functions is linear and the task becomes a linear regression task*. Since we do not know the nature of f_t, we can safely assume that our pick may not quite capture the relation between x and y. We encode this uncertainty in our pick by modeling the relation between y (i.e., f_t) and $\hat{y}$ (i.e, f_Θ) as follows

$$y = \hat{y} + \beta, \tag{1.2}$$

where β is a continuous random variable following some distribution $p(\beta)$ that expresses our uncertainty. Equation 1.2 is henceforth referred to as the *regression model equation*. This model may not be of great importance in our discussion in this chapter, but it is very important to understand the probabilistic view of the regression task, which will be left to

Assignment 1.

2 Learning f_Θ

The goal of the learning algorithm now is to find the values of the parameters w_0 and w_1 that make f_Θ as close to f_t as possible. Formally, it should minimize some error metric, i.e., $||.||_d$. One choice of error function is

$$e_u = |y_u - \hat{y}_u|^2, \tag{1.3}$$

which should be minimized over the whole training dataset (i.e., $\forall u \in \{1, \ldots, U\}$). This requirement could be expressed using the average error function

$$\mathcal{L} = \frac{1}{U}\sum_{u=1}^{U} e_u = \frac{1}{U}\sum_{u=1}^{U} |y_u - \hat{y}_u|^2 = \frac{1}{U}\sum_{u=1}^{U} |y_u - (w_1 x + w_0)|^2. \tag{1.4}$$

Equation 1.4 is called the *Mean Squared Error (MSE)* function. It measures the "loss" the algorithm incurs for any choice of parameters w_0 and w_1. Therefore, $\mathcal{L}$ is commonly referred to as the *MSE loss* in the machine learning literature. MSE loss represents the performance measure for our learning algorithm (see Lecture 1).

One might ask at this stage, how could our learning algorithm find the best choice of parameters that makes $\mathcal{L}$ as small as possible? This is a very important question, and it is the core to what is commonly known as *optimization problems*. We are not going to delve into optimization as it is a subject of its own. We will just pose our learning task in the form of an optimization problem and use some basic principles from undergraduate-level calculus to deal with it.

2.1 Learning Task Formulation

Before we introduce the algorithm that learns the parameters, we will formulate the optimization problem that describe out learning task. Given the training dataset $\mathcal{D}$, we can form

the following equation to describe the relation between all x's and $\hat{y}$'s

$$\tilde{\mathbf{y}} = \mathbf{X}\mathbf{w} \tag{1.5}$$

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ y_U \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_U \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \tag{1.6}$$

The matrix $\mathbf{X}$ is known as the *design matrix* in the literature of machine learning, not commonly used phrase for neural networks, though. An important thing to note about Equation 1.6 is that it emphasizes the role of w_0 and w_1 in the regression task; they are common to all data points, for they characterize the "line" that fits the dataset $\mathcal{D}$. Similarly and from $\mathcal{D}$, the error of all data points could be expressed as

$$\mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_U \end{bmatrix} = \begin{bmatrix} (y_1 - (w_1 x_1 + w_0))^2 \\ (y_2 - (w_1 x_2 + w_0))^2 \\ \vdots \\ (y_U - (w_1 x_U + w_0))^2 \end{bmatrix} \tag{1.7}$$

Equations 1.6 and 1.7 allow us to re-express Equation 1.4 as follow

$$\mathcal{L} = \frac{1}{U} \sum_{u=1}^{U} (y_u - (w_1 x_u + w_0))^2 \tag{1.8}$$

$$= \frac{1}{U} \left[(y_1 - (w_1 x_1 + w_0)), \ldots, (y_U - (w_1 x_U + w_0)) \right] \begin{bmatrix} (y_1 - (w_1 x_1 + w_0)) \\ \vdots \\ (y_U - (w_1 x_U + w_0)) \end{bmatrix} \tag{1.9}$$

$$= \frac{1}{U} \left(\bar{\mathbf{y}} - \mathbf{X}\mathbf{w} \right)^T \left(\bar{\mathbf{y}} - \mathbf{X}\mathbf{w} \right) \tag{1.10}$$

$$= \frac{1}{U} ||\bar{\mathbf{y}} - \mathbf{X}\mathbf{w}||_2^2, \tag{1.11}$$

where the following should be noted:

- $\bar{\mathbf{y}} = [y_1, y_2, \ldots, y_U]^T$ is a vector of all the desired responses in $\mathcal{D}$.

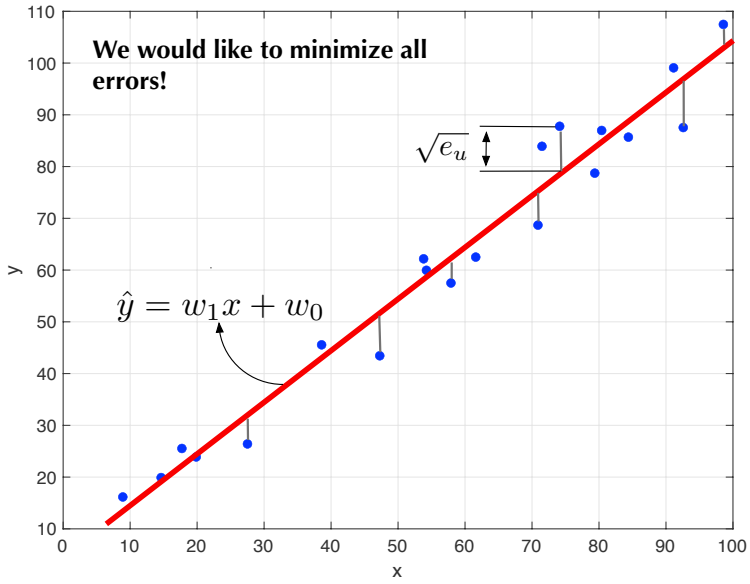


Figure 1.2

- Going from Equation 1.8 to 1.9 and from Equation 1.9 to 1.10 makes use of the definition of vector inner product in linear algebra, i.e., $\mathbf{a}^T\mathbf{a} = [a_1, \ldots, a_U][a_1, \ldots, a_U]^T = \sum_{u=1}^{U} a_u^2$.
- Going from Equation 1.10 to 1.11 makes use of the definition of the squared second norm (Euclidian distance), i.e., $||\mathbf{a}||_2^2 = \mathbf{a}^T\mathbf{a} = a_1^2 + a_2^2 + \cdots + a_U^2$.

Using Equation 1.11, we pose our learning task in the following optimization problem

$$\min_{\mathbf{w}} \frac{1}{U}||\bar{\mathbf{y}} - \mathbf{X}\mathbf{w}||_2^2. \tag{1.12}$$

The above equation should be read as follows: find the optimal vector $\mathbf{w}^\star$ from all choices of $\mathbf{w}$ that minimizes the objective function $||\bar{\mathbf{y}} - \mathbf{X}\mathbf{w}||_2^2$. Note that the objective function is a non-negative function, i.e., it takes values between 0 and ∞. This means its smallest possible value is zero, and as we get close to that value, our solution $\mathbf{w}^\star$ becomes better. Equation 1.12 is called the *linear least-squares objective*. We do not really need to understand the reason behind the name, but it is good to know it for now.

2.2 Training The Algorithm

Before we proceed with the discussion, let's try and paint a picture of what problem 1.12 aims to do. Consider Figure 1.2. These are the same points we saw in Figure 1.1, but now we have drawn a straight red line going through the points. The red line represents Equation 1.1 that our learning algorithm is trying to "fit." The loss function, which is the objective of Equation 1.12, helps do that by minimizing the average error between the data points $(x, y)_u$ and the points on the line $(x, \hat{y})_u$. As the values of w_0 and w_1 varies, the line changes. Thus, the learning algorithm's goal is to find the values that maintain the least possible error, i.e., smallest $\mathcal{L}$.

To find the best $\mathbf{w}$, also called the optimal solution $\mathbf{w}^\star$, we will recall an important principle from basic calculus, finding critical points. Remember that the objective function in problem 1.12 is non-negative and its smallest possible value is zero. Since our objective is to find $\mathbf{w}$ that minimizes $\mathcal{L}$, we will differentiate the loss with respect to (w.r.t) $\mathbf{w}$ and set the derivative to zero (do you know why?)

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 1/U \frac{\partial}{\partial \mathbf{w}} \left\{ (\bar{\mathbf{y}} - \mathbf{X}\mathbf{w})^T (\bar{\mathbf{y}} - \mathbf{X}\mathbf{w}) \right\} \tag{1.13}$$

$$= 1/U \begin{bmatrix} \frac{\partial (\bar{\mathbf{y}} - \mathbf{X}\mathbf{w})^T (\bar{\mathbf{y}} - \mathbf{X}\mathbf{w})}{\partial \mathbf{w}_0} \\ \frac{\partial (\bar{\mathbf{y}} - \mathbf{X}\mathbf{w})^T (\bar{\mathbf{y}} - \mathbf{X}\mathbf{w})}{\partial \mathbf{w}_1} \end{bmatrix} \tag{1.14}$$

$$= 1/U \left(-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{w} \right) = 0. \tag{1.15}$$

Equation 1.14 is called the *gradient* of $\mathcal{L}$ w.r.t $\mathbf{w}$, and Equation 1.15 could be derived by applying the following on Equation 1.13:

- Transpose could be distributed, i.e., $(\mathbf{a} + \mathbf{b})^T = \mathbf{a}^T + \mathbf{b}^T$.
- $\mathbf{y}^T\mathbf{X}\mathbf{w}$ is a scaler, and hence: $\mathbf{y}^T\mathbf{X}\mathbf{w} = \mathbf{w}^T\mathbf{X}^T\mathbf{y}$.
- $\mathbf{X}^T\mathbf{X}$ is a symmetric matrix, i.e., $\mathbf{X}^T\mathbf{X} = (\mathbf{X}^T\mathbf{X})^T$.
- The derivative of the quadratic form $\mathbf{w}^T\mathbf{B}\mathbf{w}$ w.r.t $\mathbf{w}$ is $(\mathbf{B} + \mathbf{B}^T)\mathbf{w}$.
- Derivative of $\mathbf{y}^T\mathbf{X}\mathbf{w}$ w.r.t $\mathbf{w}$ is $\mathbf{y}^T\mathbf{X}$.

Now, solving Equation 1.15 for $\mathbf{w}$ yields

$$\mathbf{w}^{\star} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\bar{\mathbf{y}}, \tag{1.16}$$

which includes an implicit assumption that $\mathbf{X}^T\mathbf{X}$ is full-rank matrix. Equation 1.16 is referred to as the *normal equation* or the *least squares solution* in the classical machine learning and statistical learning literatures.

2.3 REMARKS

The following should be noted:

- Linear regression problem is sometime referred to as the linear least squares approximation. This is because the algorithm (our learning algorithm) attempts to find the straight line that gives the least average squared error.
- The learning algorithm should construct the design matrix, compute $\mathbf{X}^T\mathbf{X}$, compute the inverse of $\mathbf{X}^T\mathbf{X}$ (if exists), and finally compute Equation 1.16.
- Once $\mathbf{w}$ is obtained, the algorithm is said to be trained. Plugging back $\mathbf{w}^{\star}$ into Equation 1.1 produces the machine learning model we could use to make predictions for $x \notin \mathcal{D}$.
- The discussion above focused on a 1-D observed variable, but the same solution could be derived for observed variable with multi-dimensions, i.e., $\mathbf{x} \in \mathbb{R}^N$ where $N \geq 2$. Try deriving Equation 1.16 for the following case: $\mathcal{D} = \{(\mathbf{x}, y)_u\}_{u=1}^{U}$ where $\mathbf{x} \in \mathbb{R}^3$ and $y \in \mathbb{R}$. What is the difference?
- There are other forms of linear regression that could be more powerful in learning a regression task. They employ what is called a basis function on the observed variable, i.e., $\phi(x)$. We will explore such approach in Assignment 1. Meanwhile, consider reading Chapter 3.1 in [1].

Chapter 2

Classification

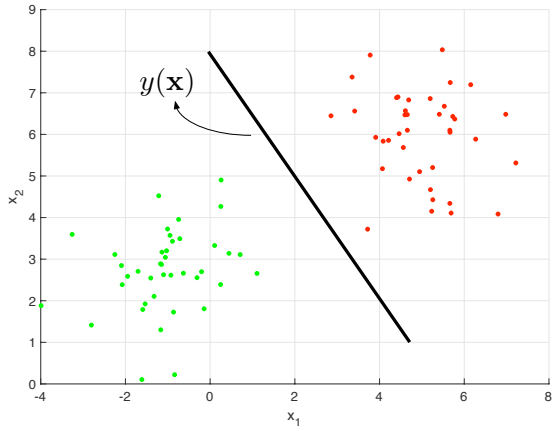
Suggested readings:
Chapter 4 of textbook [1]

In the previous chapter, we focused on the regression task, so we will turn our attention now to the other major task, which is classification. Similar to our previous treatment of regression, we will focus, here, on linear classifiers and build some foundation; however, linear classifiers are not very common in deep learning, for many practical problems require a form of non-linear classifiers.

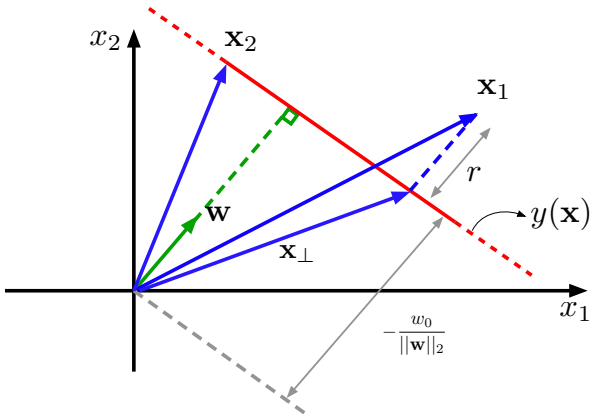
1 Linear Classifiers

Let's give a definition (a general one) for what we mean by classification.

Definition 1.1. Classification is a supervised machine learning task where a learning algorithm attempts to differentiate a number of discrete patterns referred to as the classes. The purpose of differentiating those patterns is to assign some observed variables to one of those classes. Formally, let $\mathbf{x} \in \mathbb{R}^N$ be an N-D vector of observed variables, $b \in \{1, \ldots, K\}$ is an integer representing a class label, and $f_t : \mathbf{x} \to b$ be some class-assignment function. Then, the task in classification is to learn a class-assignment function, also called a classifier, $f_\Theta : \mathbf{x} \to \hat{b}$ parameterized with a set $\Theta \in \mathbb{R}^Q$ such that $\mathcal{L}(b, \hat{b}) \leq \epsilon$, where $\mathcal{L}(.)$ defines some error metric and ϵ is some arbitrarily small non-negative scaler.



(a)



(b)

Let's take a closer look at the definition above through an example. Consider the 2-D points in Figure 2.1a (where $\mathbf{x} = [x_1\ x_2]^T$). A quick glance reveals that they represent two different colors, red, and green. Let's start by giving them some code, referred to as "label coding scheme." Red is labeled 1, and green is labeled 2, i.e., $b \in \{1, 2\}$. Now, the role of the learning algorithm is to learn the assignment-function that takes each $\mathbf{x}$ to a class label t, but how would it do that? One way is through drawing boundaries between the two classes, and assigning a point $\mathbf{x}$ to a class using its position relative to the boundary. This could be visualized in Figure 2.1a by the straight line separating the two classes. Such line is called the "decision boundary." A classifier could learn that boundary and craft a *decision rule* to assign points to classes based on that boundary.

When the decision boundary between any two classes in a classification task is made of a straight line, we refer to the classifier as a linear classifier. Such classifiers are quite interesting; they are rarely seen in real-world applications (too bad for us!), yet many complex problems could be, in one way or another, transformed to linear classification problem through *feature learning* or *feature engineering*, but again, we are getting ahead of ourselves! We will revisit this topic once again when we discuss deep learning in Part 3 of this course.

Let's go deeper and get our hands dirty with some math. The decision boundary in Figure 2.1a is merely a straight line, and from basic vector calculus, we know that a straight line

could be expressed as

$$y = [w_1, w_2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + w_0 \tag{2.1}$$

$$= \mathbf{w}^T \mathbf{x} + w_0 \tag{2.2}$$

where $\mathbf{w}_k$ is the parameter vector characterizing the line. Any point on the line results in $y(\mathbf{x}) = 0$, and hence, we can use Equation 2.1 to figure out where a point lies, i.e., $y(\mathbf{x}) \geq 0$ or $y(\mathbf{x}) \leq 0$. To illustrate this, let's look at Figure 2.1b. Point $\mathbf{x}_1$ could be assigned to class $\mathcal{C}_1$ using Equation 2.1 because

$$\mathbf{x}_1 = \mathbf{x}_\perp + r\frac{\mathbf{w}}{||\mathbf{w}||_2} \tag{2.3}$$

$$\mathbf{w}^T\mathbf{x}_1 + w_0 = \mathbf{w}^T\mathbf{x}_\perp + r\frac{\mathbf{w}^T\mathbf{w}}{||\mathbf{w}||_2} + w_0 \tag{2.4}$$

$$\mathbf{w}^T\mathbf{x}_1 + w_0 = 0 + r||\mathbf{w}||_2 \tag{2.5}$$

$$y = r||\mathbf{w}||_2 \geq 0 \tag{2.6}$$

where Equation 2.6 is a result of the facts that r is a positive integer and $||\mathbf{w}||_2$ is the second norm, which is always positive.

Generally speaking, y provides a signed value that indicates the class to which $\mathbf{x}$ belongs. This means in the case of a 2-class problem, all we need is the sign of y to determine the assignment. That could be translate into the following decision rule

$$\hat{t} = \begin{cases} 1, & y \geq 0 \\ 2, & y < 0 \end{cases} \tag{2.7}$$

The value of y itself might not be of importance in that case.

1.1 The Multi-Class Dillemma

When the number of classes is larger than 2, the sign in Equation 2.7 is not enough, and we need to figure out something else. Consider Figure 2.2. There are three classes there, and as we did before, we shall start with a coding scheme for the labels. Here, we will adopt

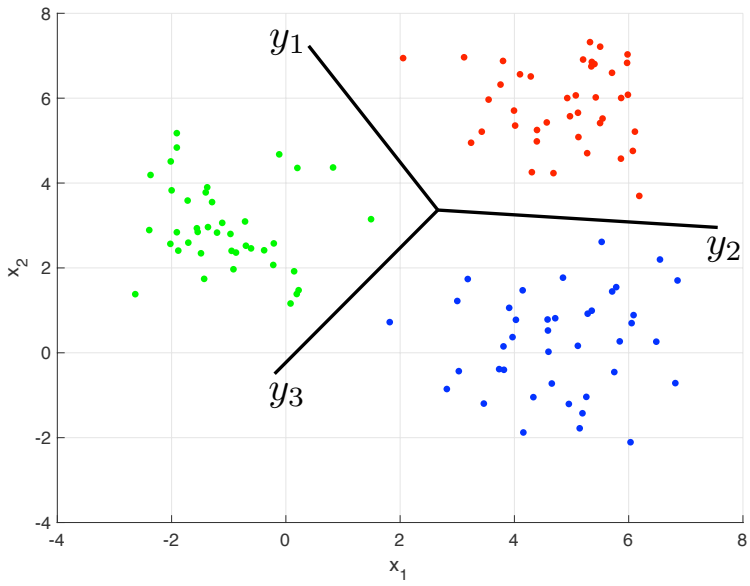


Figure 2.2

a very popular scheme that will continue with us for the rest of the course, which is the one-hot coding scheme or simply one-hot vectors. It means that t is represented with a vector $\mathbf{t} \in \{0, 1\}^K$ which only has one entry with a value of 1 where the rest is zeros. This 1 represents the class to which a point $\mathbf{x}$ belongs. Let's illustrate this with an example. The three colors in Figure 2.2 could be thought of as entries of a 3-D vector $\mathbf{t}$, where:

$$\mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \equiv \begin{bmatrix} \text{Red} \\ \text{Green} \\ \text{Blue} \end{bmatrix}. \tag{2.8}$$

For a point $\mathbf{x}_u$ in the red class, it is associated with a label vector $\mathbf{t} = [1, 0, 0]^T$. If that point is in the green class, then, $\mathbf{t} = [0, 1, 0]^T$. *It is important to note that* $\mathbf{t}$ *has to have only one entry with the value of* 1.

Different from the 2-class case, the boundary now is composed of multiple straight lines, namely y_1, y_2, and y_3. "What does that mean?" you might ask yourself at this point. This is a valid question. It means that we need to learn *three different straight lines* instead of

learning only one. Formally, this means

$$y_k = \mathbf{w}_k^T \mathbf{x} + w_{k,0}, \ \forall k \in \{1, 2, 3\} \tag{2.9}$$

where k indicates the class. We can gather all the three lines in one equation using vector notation as follows

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^T \mathbf{x} + w_{1,0} \\ \mathbf{w}_2^T \mathbf{x} + w_{2,0} \\ \mathbf{w}_3^T \mathbf{x} + w_{3,0} \end{bmatrix} \tag{2.10}$$

$$\mathbf{y} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \mathbf{w}_3^T \end{bmatrix} \mathbf{x} + \begin{bmatrix} w_{1,0} \\ w_{2,0} \\ w_{3,0} \end{bmatrix} \tag{2.11}$$

$$\mathbf{y} = \mathbf{W}^T \mathbf{x} + \mathbf{w}_0 \tag{2.12}$$

Learning three different lines, each is characterized by its own $\mathbf{w}_k$ and $w_{0,k}$, we can get the decision boundary shown in Figure 2.2. What is left to address now is the decision rule. As stated earlier, sign is not enough, but lucky us, all y's making up $\mathbf{y}$ produce signed values. Then, we can craft the following rule for decisions

$$\hat{t} = i, \quad y_i \geq y_j \quad \forall i, j \in \{1, \ldots, K\} \text{ and } i \neq j \tag{2.13}$$

2 Learning A Classifier with Least Squares

The one-million dollar question now is *how the three straight lines could be learned.* There are different ways to answer that, but in this course, we will explore two. The first one is very similar to the regression approach in Chapter 1 while the other is the perceptron algorithm. This section focuses on the regression approach, and we will call it "learning a classifier with least squares." The discussion on the perceptron algorithm will be left for the next lecture, for it paves the way for us to explore the world of neural networks and deep learning.

2.1 Task Formulation

Say that we have a training dataset $\mathcal{D} = \{(\mathbf{x}, \mathbf{t})_u\}_{u=1}^{U}$ where $\mathbf{x} \in \mathbb{R}^N$ is a vector of observed variables, and $\mathbf{t} \in \{1, 0\}^K$ is the one-hot vector indicating the class membership of $\mathbf{x}$. Our goal is to learn Equation 2.12 for K classes ($K = 3$ or some other number of classes) from $\mathcal{D}$. Then, we are going to follow similar steps to those we have followed in Chapter 1 Section 2.1. In particular, we will start by constructing a compact form for our classification problem as follows

$$\begin{bmatrix} y_{1,1} & y_{1,2} & \dots & y_{1,U} \\ y_{2,1} & y_{2,2} & \dots & y_{2,U} \\ \vdots & \vdots & \ddots & \vdots \\ y_{K,1} & y_{K,2} & \dots & y_{K,U} \end{bmatrix} = \begin{bmatrix} w_{1,0} & w_{1,1} & \dots & w_{1,N} \\ w_{2,0} & w_{2,1} & \dots & w_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K,0} & w_{K,1} & \dots & w_{K,N} \end{bmatrix} \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_{1,1} & x_{1,2} & \dots & x_{1,U} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \dots & x_{N,U} \end{bmatrix} \tag{2.14}$$

$$[\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_U]_{K \times U} = \begin{bmatrix} w_{1,0} & \mathbf{w}_1^T \\ w_{2,0} & \mathbf{w}_2^T \\ \vdots & \vdots \\ w_{K,0} & \mathbf{w}_K^T \end{bmatrix}_{K \times (N+1)} \begin{bmatrix} 1 & 1 & \dots & 1 \\ \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_U \end{bmatrix}_{(N+1) \times U} \tag{2.15}$$

$$\mathbf{Y} = \widetilde{\mathbf{W}}^T \mathbf{X} \tag{2.16}$$

Note that the main differences between Equation 2.16 and Equation 1.6 are

1. The dimensionality of the desired response $\mathbf{y}$; for regression $\mathbf{y} = y \in \mathbb{R}$ while here it is $\mathbf{y} \in \mathbb{R}^K$. This change happens because we have chosen a one-hot vector encoding for our labels.

2. The number of straight lines to learn. In regression it is a *single* line while in classification we need multiple lines—at least as many as the number of classes.

3. The dimensionality of the observed variable.

Using Equation 2.16, we can now formulate our learning task in a similar way to that we did in Chapter 1 Section 2.1. The loss function for our classification problem (i.e., posing the

learning task as an optimization problem) is

$$\min_{\mathbf{w}} \mathcal{L} = \min_{\mathbf{w}} \frac{1}{U} \sum_{u=1}^{U} ||\mathbf{t}_u - \mathbf{y}_u||_2^2 \tag{2.17}$$

The reason behind the choice of loss function in the task formulation (the objective of the optimization problem above) is rooted in the nature of the desired response. Let's try to understand that by contrasting with the task formulation in Problem 1.12. $\mathbf{t} \in \mathcal{D}$ is a one-hot vector in the classification task whereas y_u is a scalar. This makes the error definition in Equation 1.3 unsuitable now, for the error $(\mathbf{t} - \mathbf{y})$ is a vector and cannot be squared. What we need is an error metric that is: (i) a function producing a scalar even when its inputs are vectors, and (ii) a bounded quantity (e.g., non-negative). The second norm of a vector, i.e., $||\mathbf{y}||_2^2 = y_1^2 + \cdots + y_K^2$, satisfies both requirements, so we have pick it as our error metric, i.e., $e_u = ||\mathbf{t}_u - \mathbf{y}_u||_2^2$. As we did in the regression case, we seek a solution $\mathbf{w}^\star$ that minimizes the average error e_u over the training dataset $\mathcal{D}$, which is the definition of the loss in Problem 2.17 above.

Before we proceed to derive the solution, the loss of Problem 2.17 could be expressed in a more compact vector notation. This form includes all samples available in the dataset $\mathcal{D}$, and hence, it allows us to remove the summation in Problem 2.17

$$\min_{\mathbf{w}} \mathcal{L} = \min_{\mathbf{w}} \frac{1}{U} \mathrm{Tr}\left\{ (\mathbf{T} - \widetilde{\mathbf{Y}})^T (\mathbf{T} - \widetilde{\mathbf{Y}}) \right\} \tag{2.18}$$

$$= \min_{\mathbf{w}} \frac{1}{U} \mathrm{Tr}\left\{ (\mathbf{T} - \widetilde{\mathbf{W}}^T \mathbf{X})^T (\mathbf{T} - \widetilde{\mathbf{W}}^T \mathbf{X}) \right\}, \tag{2.19}$$

where Tr(.) is the *matrix trace* operator (brush up on traces from your linear algebra course!). The objective in Problem 2.19 has an interesting form, for its derivative could be easily obtained from [2].

2.2 Training The Algorithm

To train the learning algorithm, we will follow the same approach we have followed in Chapter 1 Section 2.2. Basically, we differentiate the objective of Problem 2.19 and set its derivative

to zero. Before we do so, let's expand the objective function

$$\mathcal{L} = \text{Tr}\left\{\mathbf{T}^T\mathbf{T} - \mathbf{T}^T\widetilde{\mathbf{W}}^T\mathbf{X} - \mathbf{X}^T\widetilde{\mathbf{W}}\mathbf{T} + \mathbf{X}^T\widetilde{\mathbf{W}}\widetilde{\mathbf{W}}^T\mathbf{X}\right\} \tag{2.20}$$

$$= \text{Tr}\left\{\mathbf{T}^T\mathbf{T}\right\} - \text{Tr}\left\{\mathbf{T}^T\widetilde{\mathbf{W}}^T\mathbf{X}\right\} - \text{Tr}\left\{\mathbf{X}^T\widetilde{\mathbf{W}}\mathbf{T}\right\} + \text{Tr}\left\{\mathbf{X}^T\widetilde{\mathbf{W}}\widetilde{\mathbf{W}}^T\mathbf{X}\right\}, \tag{2.21}$$

and apply the following properties

P1. $\text{Tr}(\mathbf{A}^T\mathbf{B}) = \text{Tr}(\mathbf{B}^T\mathbf{A})$ for any $K \times U$ matrices $\mathbf{A}$ and $\mathbf{B}$,

P2. $\text{Tr}(\mathbf{X}^T\widetilde{\mathbf{W}}\widetilde{\mathbf{W}}^T\mathbf{X}) = \text{Tr}(\mathbf{X}\mathbf{X}^T\widetilde{\mathbf{W}}\widetilde{\mathbf{W}}^T) = \text{Tr}(\widetilde{\mathbf{W}}^T\mathbf{X}\mathbf{X}^T\widetilde{\mathbf{W}})$, and

to obtain

$$\mathcal{L} = \text{Tr}\left\{\mathbf{T}^T\mathbf{T}\right\} - 2\text{Tr}\left\{\mathbf{X}^T\widetilde{\mathbf{W}}\mathbf{T}\right\} + \text{Tr}\left\{\mathbf{X}^T\widetilde{\mathbf{W}}\widetilde{\mathbf{W}}^T\mathbf{X}\right\}. \tag{2.22}$$

The second term of Equation 2.22 is a direct application of P1 to second and third terms of Equation 2.21 where $\mathbf{A}^T = \mathbf{T}^T$ and $\mathbf{B} = \widetilde{\mathbf{W}}^T\mathbf{X}$, and the third term of Equation 2.22 is an application of P2. From [2], we use the following derivatives

1. $\frac{\partial}{\partial \mathbf{X}}\text{Tr}(\mathbf{A}\mathbf{X}\mathbf{B}) = \mathbf{A}^T\mathbf{B}^T$.
2. $\frac{\partial}{\partial \mathbf{X}}\text{Tr}(\mathbf{X}^T\mathbf{B}\mathbf{X}) = \mathbf{B}\mathbf{X} + \mathbf{B}^T\mathbf{X}$

to differentiate $\mathcal{L}$ w.r.t. $\widetilde{\mathbf{W}}$ and set it to zero

$$\frac{\partial \mathcal{L}}{\partial \widetilde{\mathbf{W}}} = -2\mathbf{X}\mathbf{T}^T + 2\mathbf{X}\mathbf{X}^T\widetilde{\mathbf{W}} = 0. \tag{2.23}$$

Solving for $\widetilde{\mathbf{W}}$ yields

$$\widetilde{\mathbf{W}}^\star = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{T}^T \tag{2.24}$$

This solution is quite similar to the that we have obtained in Chapter 1, which is why we will keep calling it the *the least squares solution* and *normal equation*.

Bibliography

[1] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[2] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," nov 2012, version 20121115. [Online]. Available: http://www2.compute.dtu.dk/pubdb/pubs/3274-full.html