

Class #05:  
Linear Classification

Machine Learning (COMP 135): M. Allen, 03 Feb. 20

1

## Review: The General Learning Problem

- ▶ We want to learn functions from inputs to outputs, where each input has  $n$  features:

Inputs  $\langle x_1, x_2, \dots, x_n \rangle$ , with each feature  $x_i$  from domain  $X_i$ .  
Outputs  $y$  from domain  $Y$ .

Function to learn:  $f : X_1 \times X_2 \times \dots \times X_n \rightarrow Y$

- ▶ The type of learning problem we are solving really depends upon the type of the output domain,  $Y$ 
  1. If output  $Y \in \mathbb{R}$  (a real number), this is **regression**
  2. If output  $Y$  is a finite discrete set, this is **classification**

▶ Monday, 3 Feb. 2020
Machine Learning (COMP 135) 2

2

## Decisions to Make

- ▶ When collecting our training example pairs,  $(x, f(x))$ , we still have some decisions to make
- ▶ **Example:** Medical Informatics
  - ▶ We have some genetic information about patients
  - ▶ Some get sick with a disease and some don't
  - ▶ Patients live for a number of years (sick or not)
- ▶ **Question:** what do we want to learn from this data?
- ▶ Depending upon what we decide, we may use:
  - ▶ Different models of the data
  - ▶ Different machine learning approaches
  - ▶ Different measurements of successful learning

▶ Monday, 3 Feb. 2020
Machine Learning (COMP 135) 3

3

## One Approach: Regression

- ▶ We decide that we want to try to learn to predict how long patients will live
- ▶ We base this upon information about the degree to which they express a specific gene
- ▶ A **regression problem**: the function we learn is the "best (linear) fit" to the data we have

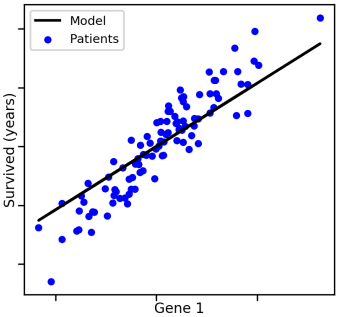


Image source: <https://aldro61.github.io/microbiome-summer-school-2017/sections/basics/>

▶ Monday, 3 Feb. 2020
Machine Learning (COMP 135) 4

4

## Another Approach: Classification

- ▶ We decide instead that we simply want to decide whether a patient will get the disease or not
- ▶ We base this upon information about expression of two genes
- ▶ A **classification problem**: learned function separates individuals into 2 groups (binary classes)

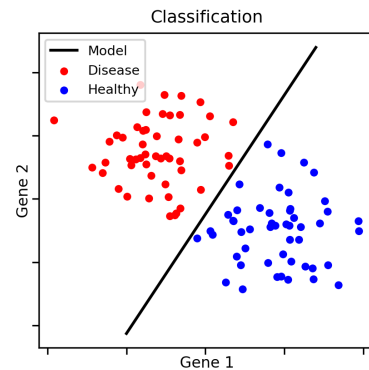


Image source: <https://aldro61.github.io/microbiome-summer-school-2017/sections/basics/>

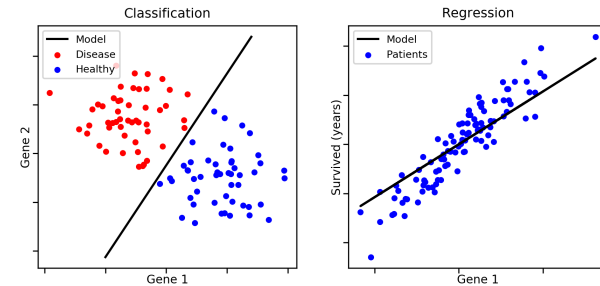
▶ Monday, 3 Feb. 2020

Machine Learning (COMP 135)

5

5

## Which is the Correct Approach?



- ▶ The approach we use depends upon what we want to achieve, and what works best based upon the data we have
- ▶ Much machine learning involves investigating different approaches

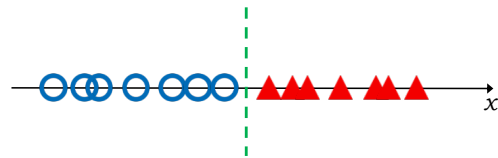
▶ Monday, 3 Feb. 2020

Machine Learning (COMP 135)

6

6

## From Regression to Classification



- ▶ Suppose we have two classes of data, defined by a single attribute  $x$
- ▶ We seek a **decision boundary** that splits the data in two
- ▶ When such a boundary can be defined using a linear function, it is called a **linear separator**

▶ Monday, 3 Feb. 2020

Machine Learning (COMP 135)

7

7

## Threshold Functions

1. We have data-points with  $n$  features:  

$$\mathbf{x} = (x_1, x_2, \dots, x_n)$$
2. We have a linear function defined by  $n+1$  weights:  

$$\mathbf{w} = (w_0, w_1, w_2, \dots, w_n)$$
3. We can write this linear function as:  

$$\mathbf{w} \cdot \mathbf{x}$$
4. We can then find the **linear boundary**, where:  

$$\mathbf{w} \cdot \mathbf{x} = 0$$
5. And use it to define our **threshold** between classes:

$$h_{\mathbf{w}} = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \mathbf{w} \cdot \mathbf{x} < 0 \end{cases}$$

Outputs 1 and 0 here are **arbitrary labels** for one of two possible classes

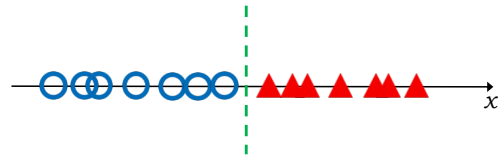
▶ Monday, 3 Feb. 2020

Machine Learning (COMP 135)

8

8

## From Regression to Classification



- ▶ Data is **linearly separable** if it can be divided into classes using a linear boundary:

$$\mathbf{w} \cdot \mathbf{x} = 0$$

- ▶ Such a boundary, in 1-dimensional space, is a **threshold value**

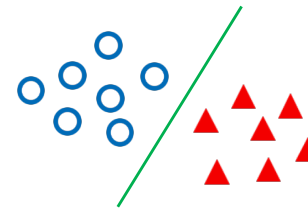
Monday, 3 Feb. 2020

Machine Learning (COMP 135)

9

9

## From Regression to Classification



- ▶ Data is **linearly separable** if it can be divided into classes using a linear boundary:

$$\mathbf{w} \cdot \mathbf{x} = 0$$

- ▶ Such a boundary, in 2-dimensional space, is a **line**

Monday, 3 Feb. 2020

Machine Learning (COMP 135)

10

10

## From Regression to Classification

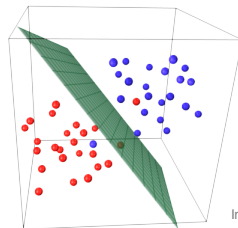


Image: R. Urtasun (U. of Toronto)

- ▶ Data is **linearly separable** if it can be divided into classes using a linear boundary:

$$\mathbf{w} \cdot \mathbf{x} = 0$$

- ▶ Such a boundary, in 3-dimensional space, is a **plane**
- ▶ In higher dimensions, it is a **hyper-plane**

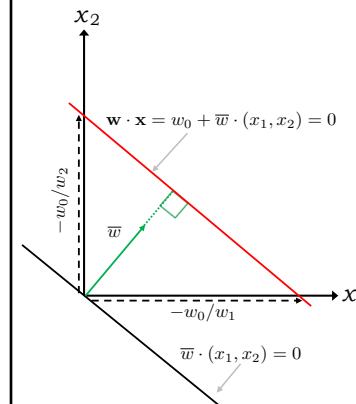
Monday, 3 Feb. 2020

Machine Learning (COMP 135)

11

11

## The Geometry of Linear Boundaries



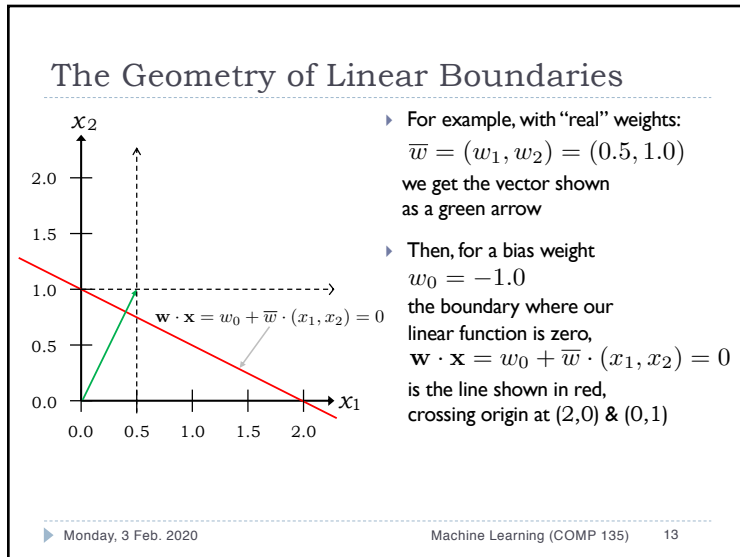
- ▶ Suppose we have 2-dimensional inputs  $\mathbf{x} = (x_1, x_2)$
- ▶ The “real” weights  $\bar{\mathbf{w}} = (w_1, w_2)$  define a **vector**
- ▶ The boundary where our linear function is zero,  $\mathbf{w} \cdot \mathbf{x} = w_0 + \bar{\mathbf{w}} \cdot (x_1, x_2) = 0$  is an orthogonal line, parallel to  $\bar{\mathbf{w}} \cdot (x_1, x_2) = 0$
- ▶ Its offset from origin is determined by  $w_0$  (which is called the **bias weight**)

Monday, 3 Feb. 2020

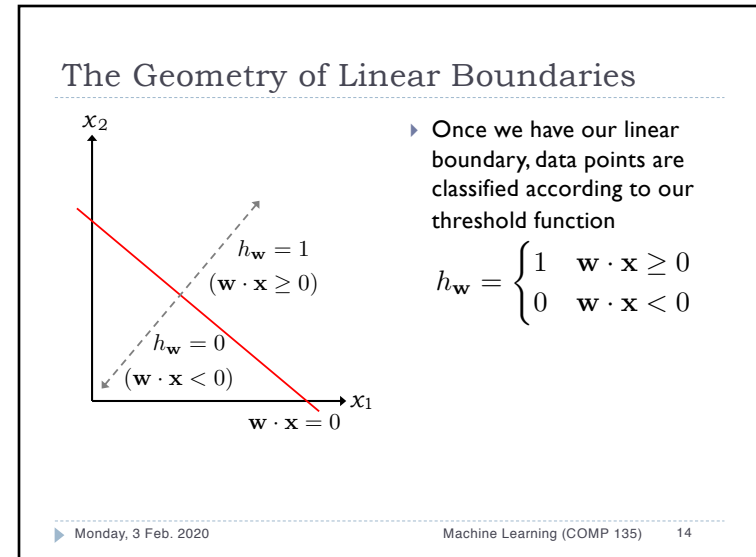
Machine Learning (COMP 135)

12

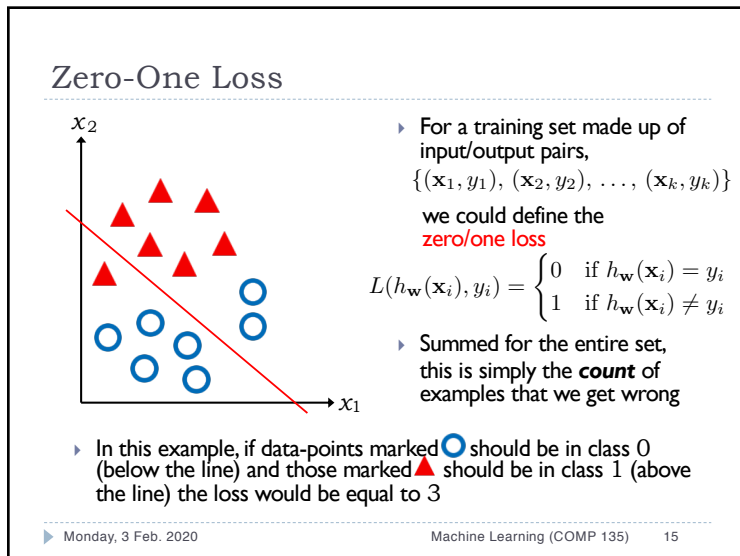
12



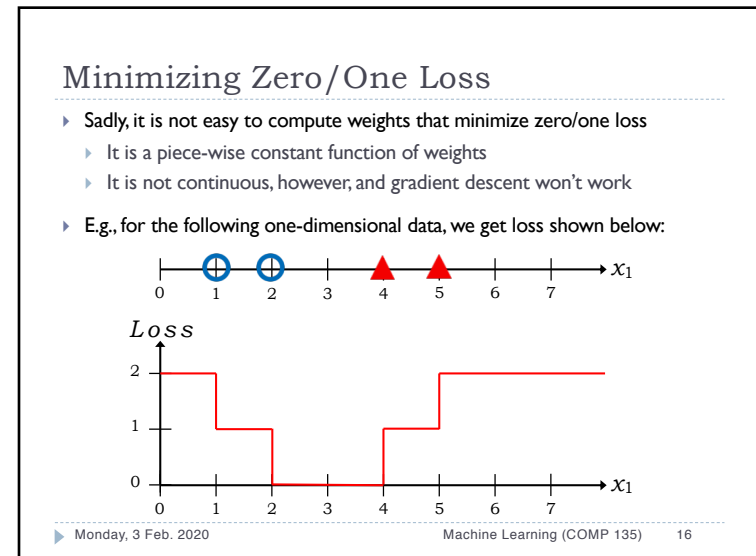
13



14



15



16

## Perceptron Loss

- ▶ Instead, we define the **perceptron loss** on a training item:

$$\mathbf{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,n})$$

$$L_\pi(h_{\mathbf{w}}(\mathbf{x}_i), y_i) = \sum_{j=0}^n (y_i - h_{\mathbf{w}}(\mathbf{x}_i)) \times x_{i,j}$$

- ▶ For example, suppose we have a 2-dimensional element in our training set for which the correct output is 0, but our threshold function says 1:

$$\mathbf{x}_i = (0.5, 0.4) \quad y_i = 1 \quad h_{\mathbf{w}}(\mathbf{x}_i) = 0$$

$$L_\pi(h_{\mathbf{w}}(\mathbf{x}_i), y_i) = (1 - 0)(1 + 0.5 + 0.4) = 1.9$$

The difference between what output **should** be, and what our weights make it

Sum of input attributes (1 is the “dummy” attribute that is multiplied by bias weight  $w_0$ )

▶ Monday, 3 Feb. 2020

Machine Learning (COMP 135) 17

17

## Perceptron Learning

- ▶ To minimize perceptron loss we can start from initial weights—perhaps chosen uniformly from interval  $[-1, 1]$ —and then:

1. Choose an input  $\mathbf{x}_i$  from our data set that is wrongly classified.
2. Update vector of weights,  $\mathbf{w} = (w_0, w_1, w_2, \dots, w_n)$ , as follows:

$$w_j \leftarrow w_j + \alpha(y_i - h_{\mathbf{w}}(\mathbf{x}_i)) \times x_{i,j}$$

3. Repeat until no classification errors remain.

- ▶ The update equation means that:

1. If correct output should be *below* the boundary ( $y_i = 0$ ) but our threshold has placed it *above* ( $h_{\mathbf{w}}(\mathbf{x}_i) = 1$ ) then we *subtract* each feature ( $x_{i,j}$ ) from the corresponding weight ( $w_i$ )
2. If correct output should be *above* the boundary ( $y_i = 1$ ) but our threshold has placed it *below* ( $h_{\mathbf{w}}(\mathbf{x}_i) = 0$ ) then we *add* each feature ( $x_{i,j}$ ) to the corresponding weight ( $w_i$ )

▶ Monday, 3 Feb. 2020

Machine Learning (COMP 135) 18

18

## Perceptron Updates

$$w_j \leftarrow w_j + \alpha(y - h_{\mathbf{w}}(\mathbf{x}_i)) \times x_{i,j}$$

- ▶ The perceptron update rule shifts the weight vector positively or negatively, trying to get all data on the right side of the linear decision boundary

- ▶ Again, supposing we have an error as before, with weights as given below:

$$\mathbf{x}_i = (0.5, 0.4) \quad \mathbf{w} = (0.2, -2.5, 0.6) \quad y_i = 1$$

$$\mathbf{w} \cdot \mathbf{x}_i = 0.2 + (-2.5 \times 0.5) + (0.6 \times 0.4) = -0.81 \quad h_{\mathbf{w}}(\mathbf{x}_i) = 0$$

- ▶ This means we add the value of each attribute to its matching weight (assuming again that “dummy”  $x_{i,0} = 1$ , and that parameter  $\alpha = 1$ ):

$$w_0 \leftarrow (w_0 + x_{i,0}) = (0.2 + 1) = 1.2$$

$$w_1 \leftarrow (w_1 + x_{i,1}) = (-2.5 + 0.5) = -2.0$$

$$w_2 \leftarrow (w_2 + x_{i,2}) = (0.6 + 0.4) = 1.0$$

$$\mathbf{w} \cdot \mathbf{x}_i = 1.2 + (-2.0 \times 0.5) + (1.0 \times 0.4) = 0.6 \quad h_{\mathbf{w}}(\mathbf{x}_i) = 1$$

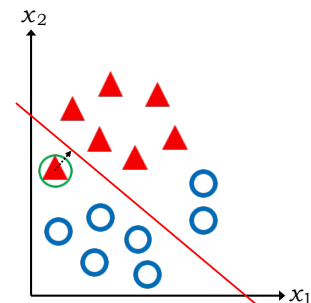
After adjusting weights, our function is now correct on this input

▶ Monday, 3 Feb. 2020

Machine Learning (COMP 135) 19

19

## Progress of Perceptron Learning



- ▶ For an example like this, we:

1. Choose a mis-classified item (marked in green)
2. Compute the weight updates, based on the “distance” away from the boundary (so weights shift more based upon errors in boundary placement that are more extreme)

- ▶ Here, this **adds** to each weight, changing the decision boundary

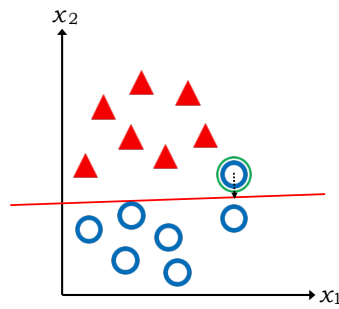
- ▶ In this example, data-points marked  $\circ$  should be in class 0 (below the line) and those marked  $\triangle$  should be in class 1 (above the line)

▶ Monday, 3 Feb. 2020

Machine Learning (COMP 135) 20

20

## Progress of Perceptron Learning



- ▶ Once we get a new boundary, we repeat the process
- 1. Choose a mis-classified item (marked in green)
- 2. Compute the weight updates, based on the “distance” away from the boundary (so weights shift more based upon errors in boundary placement that are more extreme)
- ▶ Here, this **subtracts** from each weight, changing the decision boundary in the other direction

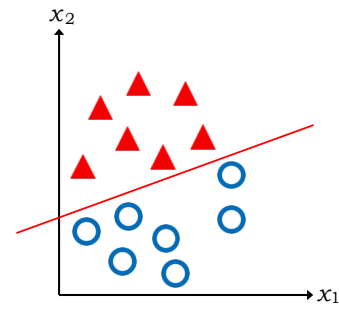
▶ In this example, data-points marked ● should be in class 0 (below the line) and those marked ▲ should be in class 1 (above the line)

▶ Monday, 3 Feb. 2020

Machine Learning (COMP 135) 21

21

## Linear Separability



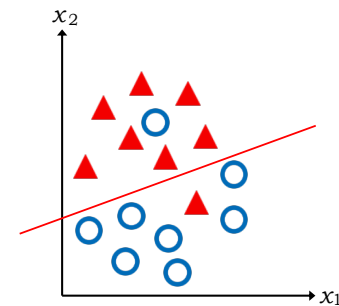
- ▶ The process of adjusting weights stops when there is no classification error left
- ▶ A data-set is **linearly separable** if a linear separator exists for which there will be no error
- ▶ It is possible that there are **multiple** linear boundaries that achieve this
- ▶ It is also possible that there is **no such** boundary!

▶ Monday, 3 Feb. 2020

Machine Learning (COMP 135) 22

22

## Linearly Inseparable Data



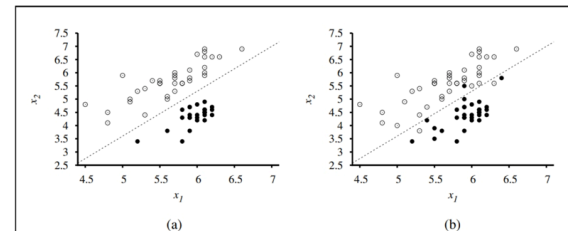
- ▶ Some data **can't** be separated using a linear classifier
  - ▶ Any line drawn will always leave some error
- ▶ The perceptron update method is guaranteed to eventually **converge** to an error-free boundary **if** such a boundary really exists
  - ▶ If it **doesn't** exist, then the most basic version of the algorithm will never terminate

▶ Monday, 3 Feb. 2020

Machine Learning (COMP 135) 23

23

## Linearly Inseparable Data



**Figure 18.15** (a) Plot of two seismic data parameters, body wave magnitude  $x_1$  and surface wave magnitude  $x_2$ , for earthquakes (white circles) and nuclear explosions (black circles) occurring between 1982 and 1990 in Asia and the Middle East (Kebeasy *et al.*, 1998). Also shown is a decision boundary between the classes. (b) The same domain with more data points. The earthquakes and explosions are no longer linearly separable.

Image source: Russel & Norvig, *AI: A Modern Approach* (Prentice Hal, 2010)

- ▶ Unfortunately, data that can't be separated linearly is very common...

▶ Monday, 3 Feb. 2020

Machine Learning (COMP 135) 24

24

## Modifying Perceptron Learning

► To minimize error, we can modify the algorithm slightly:

1. Choose an input  $\mathbf{x}_i$  from our data set that is wrongly classified.
2. Update vector of weights,  $\mathbf{w} = (w_0, w_1, w_2, \dots, w_n)$ , as follows:

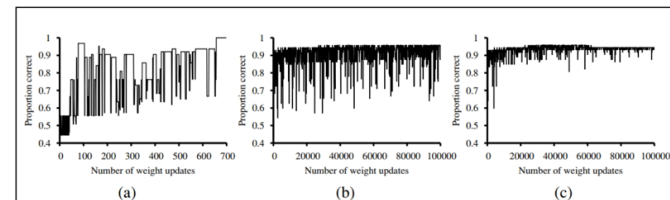
$$w_j \leftarrow w_j + \alpha(y_i - h_{\mathbf{w}}(\mathbf{x}_i)) \times x_{i,j}$$

~~3. Repeat until no classification errors remain.~~

3. Repeat until weights no longer change; modify learning parameter  $\alpha$  over time to guarantee this.

- If we make  $\alpha$  smaller and smaller over time, then as  $\alpha \rightarrow 0$ , the weights will quit changing, and the algorithm converges
- To get down to a **least-error** possible final separator, we do this slowly, e.g., setting  $\alpha(t) = 1000/(1000 + t)$ , where  $t$  is the current iteration of the update algorithm

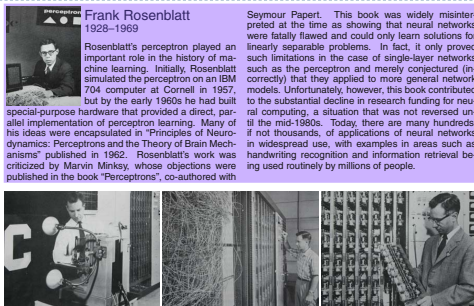
## Modifying Perceptron Learning



**Figure 18.16** (a) Plot of total training-set accuracy vs. number of iterations through the training set for the perceptron learning rule, given the earthquake/explosion data in Figure 18.15(a). (b) The same plot for the noisy, non-separable data in Figure 18.15(b); note the change in scale of the  $x$ -axis. (c) The same plot as in (b), with a learning rate schedule  $\alpha(t) = 1000/(1000 + t)$ .

Image source: Russel & Norvig, *AI: A Modern Approach* (Prentice Hall, 2010)

## The History of the Perceptron



**Figure 4.8** Illustration of the Mark 1 perceptron hardware. The photograph on the left shows how the inputs were obtained using a simple camera system in which an input scene, in this case a printed character, was illuminated by powerful lights, and an image focused onto a  $20 \times 20$  array of cadmium sulfide photocells, giving a primitive 400 pixel image. The perceptron also had a patch board, shown in the middle photograph, which allowed different configurations of input features to be tried. Often these were wired up at random to demonstrate the ability of the perceptron to learn without the need for precise wiring, in contrast to a modern digital computer. The photograph on the right shows one of the racks of adaptive weights. Each weight was implemented using a rotary variable resistor, also called a potentiometer, driven by an electric motor thereby allowing the value of the weight to be adjusted automatically by the learning algorithm.

From: C. Bishop, *Pattern Recognition and Machine Learning*, Springer (2006).

## This Week

► Linear classification; evaluating algorithms

► Readings:

- Book excerpts on linear methods and evaluation metrics
  - Posted to Piazza, linked from class schedule

► Assignment 02: due Wednesday, 12 Feb.

► Office Hours: 237 Halligan

- Monday, Noon – 1:30 PM
- Tuesday, 9:00 AM – 10:30 AM