

# Machine Learning (Linear Classifiers and Regression)

Prof K R Chowdhary

MBM University

October 25, 2024



# Linear Classifiers

⇒ Examples with  $n$ -dimensional instance space, positive and negative examples tend to cluster in different regions.

⇒ This observation motivates us to use another approach to classification where we identify decision surfaces that separates the two classes.

⇒ A very simple approach is to use a linear function.

⇒ The goal of predictive modeling is to build a model that predicts some specified attribute(s) value from the values of the other attributes.

⇒ We will elaborate on linear classifier in general.

⇒ We shall use a domain with attributes as real numbers. To use these attributes in a algebraic function shown in Fig. 1



# Linear Classifier

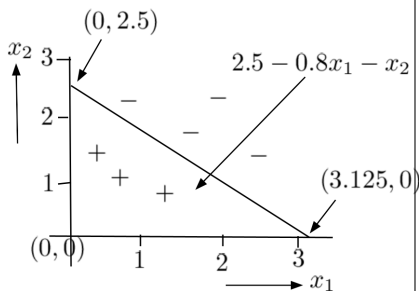


Figure 1: A linear classifier in a domain of two real valued attributes  $x_1, x_2$

⇒ Examples are labeled as

<sup>1</sup>This equation is a standard line equation  $y = mx + c$ , where  $m$  is slope and  $c$  is point where this line intersects on  $x$  axis. We have used coordinates  $x_1, x_2$ , which can be extended to  $n$  coordinates  $x_1 \dots x_n$

positive (+) and negative (-), and two classes are separated by a linear function:

$$2.5 - 0.8x_1 - x_2 = 0 \quad (1)$$

⇒ Equation (1)<sup>1</sup>, the variables  $x_1$  and  $x_2$  are real numbers.

Exercise: Given the graph in Fig. (1), construct the eq. (1).

Hint: Extend line to  $-x_1$  direction, and the angle  $m$  in  $y = mx + c$  is  $-\frac{2}{2.5} = 0.8$



⇒ Table 1 shows seven examples of attributes  $(x_1, x_2)$ , value of classifier “ $2.5 - 0.8x_1 - x_2$ ,” and class of the example.

⇒ When value of classifier is *neg*, point or the coordinate is falling above the straight line in Fig. 1, when classifier returns positive, the example is taken as belonging to pos class.

⇒ Hence, given a classifier like this, we are able to classify any

attribute set, which is a two dimensional vector.

**Table 1:** Set of attributes  $(x_1, x_2)$  and their classes

$x_1$	$x_2$	$2.5 - 0.8x_1 - x_2$	Class
1.0	2.3	-0.6	neg
1.6	1.8	-0.58	neg
2.1	2.7	-1.88	neg
2.4	1.4	-0.82	neg
0.8	1.1	+0.76	pos
0.8	1.8	+0.06	pos
1.4	0.8	+0.58	pos



# Linear Classifier...

⇒ Note only the linear classifier (1) classifies examples as *pos* and *neg*, but any classifier, e.g.,  $1.5 + 2.1x_1 - 1.1x_2$ , will classify infinitely large number of examples as *pos/neg*. Generic form is:

$$w_0 + w_1x_1 + w_2x_2 = 0. \quad (2)$$

And, for a domains with  $n$  attributes is:

$$w_0 + w_1x_1 + \dots + w_nx_n = 0. \quad (3)$$

In eq. (3), if  $n = 2$ , it a line, if  $n = 3$ , it is a plane, for  $n > 3$ , it

is a *hyperplane*. If 0th attribute  $x_0 = 1$ , eq. (3) becomes:

$$\sum_{i=0}^n w_ix_i = 0. \quad (4)$$

Classifier's behavior is decided by coefficients  $w_i$  (weights). Task of ML is: find out  $w_i$ 's values. In equ.  $y = mx + c$ , the  $m$  is angle w.r.t.  $x$  axis, and in (3), coefficients  $w_1, \dots, w_n$  define angle of hyperplane, w.r.t. system of coordinates,  $w_0$  is *bias* or *offset* – the hyperplane has distance from the system coordinates.



⇒ *Bias versus Threshold*: Bias is amount of error introduced by approximating real-world phenomena with a simplified model.

⇒ *Bias* in Fig. 1 is  $w_0 = 2.5$ , lower the bias, classifier shifts closer to origin  $[0, 0]$ , higher value shifts it away from origin. At,  $w_0 = 0$ , the classifier intersects the origin of the coordinate system. Equation (3) can also be written as:

$$w_1x_1 + w_2x_2 + \dots + w_nx_n = \theta, \quad (5)$$

here,  $\theta = -w_0$ . This  $\theta$  is called *threshold* that weighted sum has to exceed it, if the example is to be positive.

Table 2: Attributes  $(x_1, x_2)$ , their weighted sum and threshold

$x_1$	$x_2$	$(-0.8x_1 - x_2)$	$\theta$
1.0	2.3	-3.3	-2.5
1.6	1.8	-3.8	-2.5
2.1	2.7	-4.26	-2.5
2.4	1.4	-3.52	-2.5
0.8	1.1	-1.74	-2.5
0.8	1.8	-2.24	-2.5
1.4	0.8	-1.92	-2.5



⇒ Last 3 examples (table 2): weighted sum in third column exceeds  $\theta$ , so they have *pos* labels. First 4 examples: weighted sum  $< \theta$ , so label=*neg*.

⇒ *Perceptron Learning*: To simplify linear classifier, we assume that training example  $\mathbf{x}$  is described by  $n$  binary attributes for  $n$  dimensions,  $x_i \in \mathbf{x}$  is binary, i.e., 0 or 1.

⇒ For  $c(\mathbf{x}) = 1$ , class=*pos*, for  $c(\mathbf{x}) = 0$ , it is *neg*. Real class is  $c$ , and hypothesized class is

$h(\mathbf{x})$ , ( $h =$  hypothesis). If,  $\sum_{i=0}^n w_i x_i > 0$ , classifier hypothesizes  $\mathbf{x}$  as *pos*, so  $h(\mathbf{x}) = 1$ .

⇒ When,  $\sum_{i=0}^n w_i x_i \leq 0$ , label is *neg* and  $h(\mathbf{x}) = 0$ .

⇒ Examples with  $c(\mathbf{x}) = 1$  are linearly separable from those with  $c(\mathbf{x}) = 0$ . So, there exists a linear classifier that can label correctly all the training examples  $\mathbf{x}$ , and for each  $h(\mathbf{x}) = c(\mathbf{x})$ . Task of ML: find weights  $w_i$  that correctly classifies all  $\mathbf{x}$ .



# Inducing the Linear Classifier

⇒ Objective: for any attribute example  $\mathbf{x}$  with real class  $c(\mathbf{x}) = 1$ , the classifier must hypothesize the example as positive, i.e.  $h(\mathbf{x}) = 1$ , and when  $c(\mathbf{x}) = 0$ , it must hypothesize  $\mathbf{x}$  as negative, i.e.  $h(\mathbf{x}) = 0$ . We can do this by adjusting the weight  $w_i$ .

⇒ When classifier is presented with training example  $\mathbf{x}$ , it must return its label as  $h(\mathbf{x})$ . If  $c(\mathbf{x}) \neq h(\mathbf{x})$ , weights  $w_i$  are not perfect, so they must be modified so that  $c(\mathbf{x}) = h(\mathbf{x})$ .

⇒ Assume that  $c(\mathbf{x}) = 1$  and

$h(\mathbf{x}) = 0$ . This happens only if  $\sum_{i=0}^n w_i x_i < 0$ : an indication that the weights are too small. So, weights must be increased so that  $\sum_{i=0}^n w_i x_i > 0$ , (So,  $h(\mathbf{x}) = 1$ ).

⇒ It is simple to understand that only the weight of  $w_i$  be increased for which  $x_i = 1$ , (when  $x_i = 0$ ,  $w_i \cdot x_i = w_i \cdot 0 = 0$ ). (This is the reason for choosing binary attributes!!).

⇒ Similarly, when  $c(\mathbf{x}) = 0$  and  $h(\mathbf{x}) = 1$ , we decrease the weights  $w_i$  for which  $x_i$  are 1, so that  $\sum_{i=0}^n w_i x_i \leq 0$ .



# Weight adjustment in Perceptron

*Weight adjustment Summary:*

- Hypothesized label,  $h(\mathbf{x}) = 1$  when real class label  $c(\mathbf{x}) = 0$ : decrease  $w_i$  for attribute  $x_i = 1$ ,
- Hypothesized label,  $h(\mathbf{x}) = 0$  when real class label  $c(\mathbf{x}) = 1$ : increase  $w_i$  for attribute  $x_i = 1$ ,
- Both labels same,  $c(\mathbf{x}) = h(\mathbf{x})$ : no wt. adjustment reqd.

Regulate the weights by:

$$w_i = w_i + \eta \cdot [c(\mathbf{x}) - h(\mathbf{x})] \cdot x_i \quad (6)$$

$\eta \in (0, 1]$ , called *learning rate*.

- Checking validity of equation (6): (i) When  $c(\mathbf{x}) = h(\mathbf{x})$ :  $w_i$  remains unchanged.

(ii) When  $c(\mathbf{x}) = 1$  and  $h(\mathbf{x}) = 0$ : RHS of equ. (6) is:  $w_i + \eta \cdot 1 \cdot x_i = w_i + \eta$ , as  $x_i = 1$ . This increases  $w_i$ , so it is  $\geq 1$ , hence perceptron fires, and makes  $h(\mathbf{x}) = 1$ .

(iii) When  $c(\mathbf{x}) = 0$  but  $h(\mathbf{x}) = 1$ : RHS of equ. (6) is:  $w_i + \eta \cdot [-1] \cdot 1 = w_i - \eta$ , as  $x_i = 1$ . This decreases  $w_i$  to  $\leq 0$ , it stops the perceptron from firing, and makes  $h(\mathbf{x}) = 0$ .

This concludes how perceptron hypothesizes the same label as the label of  $c(\mathbf{x})$ .



# Perceptron Learning Algorithm

⇒ To start with, weights  $w_i$  of perceptron are initialized to some random values. Next, each training example  $\mathbf{x}$  with attributes  $x_1, \dots, x_i, \dots, x_n$ , is presented to the classifier, one at a time. Each time, every weight of the classifier is subjected to equation (6).

⇒ The training for last example  $\mathbf{x}$  shows that one *epoch* (round)

of training is complete. If all the labels are correctly hypothesized, indicated by  $h(\mathbf{x}) = c(\mathbf{x})$ , the training process is terminated, else it repeats from first example again. Usually, many such rounds are needed to train the perceptron. The corresponding algorithm is shown as algorithm 1.



---

## Algorithm 1 Perceptron learning Algorithm

---

- 1: % Let two classes be  $c(\mathbf{x}) = 1$  and  $c(\mathbf{x}) = 0$ , and they are linearly separable.
  - 2: Initialize weights  $w_i$  to some small random numbers.
  - 3: Choose some suitable learning rate  $\eta \in (0, 1]$ .
  - 4: **while**  $c(\mathbf{x}) \neq h(\mathbf{x})$  for all training examples **do**
  - 5:   **for** each training example  $\mathbf{x} = (x_1, \dots, x_n)$ , having class  $c(\mathbf{x})$  **do**
  - 6:      $h(\mathbf{x}) = 1$  if  $\sum_{i=0}^n w_i x_i > 0$ , otherwise  $h(\mathbf{x}) = 0$ .
  - 7:     Update each weight using the formula, (6)
  - 8:   **end for**
  - 9: **end while**
- 



# Example on Perceptron Learning Algorithm

We are given a table of examples as 3, with three examples Ex1 to Ex3, each having three binary attributes.

**Table 3:** Examples for perceptron learning

Example	$x_1$	$x_2$	$x_3$	$c(\mathbf{x})$
Ex1	1	1	0	1
Ex2	0	0	1	0
Ex3	1	0	1	0

We consider that learning rate  $\eta = 0.6$ , and randomly

generated initial weights

$w_0, w_1, w_2, w_3$  are

$[0.15, 0.2, 0.1, 0.25]$  and  $x_0 = 1$ .

Given these, our objective is to separate the “+” examples (Ex1) from “-” examples (Ex2, Ex3).

The classifier's hypothesis about class  $\mathbf{x}$ :  $h(\mathbf{x}) = 1$  if

$\sum_{i=0}^n w_i x_i > 0$ , and  $h(\mathbf{x}) = 0$ ,

otherwise. After each example

is presented to the classifier, all the weights are adjusted through formula (6), as table 5 shows.



# Example on Perceptron Learning Algorithm...

Table 4: Weight adjustments for perceptron learning

Var. →	$x_1$	$x_2$	$x_3$	$w_0$	$w_1$	$w_2$	$w_3$	$c(\mathbf{x})$	$h(\mathbf{x})$	$c(\mathbf{x})-h(\mathbf{x})$
Examples ↓										
Random classifier				0.15	0.2	0.1	0.25			
Ex1 →	1	1	0					1	1	0
New Classifier:				0.15	0.2	0.1	0.25			
Ex2 →	0	0	1					0	1	-1
New Classifier:				-0.45	0.8	0.7	-0.35			
Ex3 →	1	0	1					0	0	0
New Classifier:				-0.45	0.8	0.7	-0.35			



## Example on Perceptron Learning Algorithm...

⇒ Let us see how the computation in table 5 are computed: First, we find out hypothesized class  $h(\mathbf{x})$  in Ex1:

$$\begin{aligned} & \sum_{i=0}^n w_i x_i \\ &= w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 \\ &= 0.15 \times 1 + 0.2 \times 1 + 0.1 \times 1 + 0.25 \times 0 \\ &= 0.45 \end{aligned}$$

Hence,  $h(\mathbf{x}) = 1$ . Since,  $c(\mathbf{x}) = 1$  and  $h(\mathbf{x}) = 1$ , their difference is zero. So, no weight will change, and the final weights remains the same as it is for Ex2.

⇒ Since  $c(\mathbf{x}) - h(\mathbf{x}) \neq 0$  in Ex1 to Ex3, we make 2nd round, where weights of Ex3 are used in place of Random classifier at top of the table, and calculate the table again.

⇒ This also does not make  $c(\mathbf{x}) = h(\mathbf{x})$ , we make one more round resulting table is given in next slide.



# Example on Perceptron Learning Algorithm...

Table 5: Weight adjustments for perceptron learning

Var. →	$x_1$	$x_2$	$x_3$	$w_0$	$w_1$	$w_2$	$w_3$	$c(\mathbf{x})$	$h(\mathbf{x})$	$c(\mathbf{x})-h(\mathbf{x})$
Examples ↓										
Random classifier				-0.45	0.2	0.7	-0.95			
Ex1 →	1	1	0					1	1	0
New Classifier:				-0.45	0.2	0.7	-0.95			
Ex2 →	0	0	1					0	0	0
New Classifier:				-0.45	0.2	0.7	-0.95			
Ex3 →	1	0	1					0	0	0
New Classifier:				-0.45	0.2	0.7	-0.95			



Conclusion: Perceptron Learning Algorithm  $\Rightarrow$  So, the classifier is trained in three steps. It classifies Ex1 in one class (+) and Ex2, Ex3 in “-” class, as seen in the column  $c(\mathbf{x}) - h(\mathbf{x})$  of the table. Final version of classifier:

$-0.45 + 0.2x_1 + 0.7x_2 - 0.95x_3 = 0$ , no longer classifies wrongly. The training has thus been completed in three epochs (rounds).

$\Rightarrow$  Note: Learning has taken place using perceptron, and have obtained a linear classifier:

$$-0.45 + 0.2x_1 + 0.7x_2 - 0.95x_3 = 0, \quad (7)$$

Now, it classify any amount of data as either “+” or “-”, in single step. So, classifier has been *induced*.

$\Rightarrow$  Important: Irrespective of the initial weights ( $w_0..w_n$ ), size  $n$  of attribute vector, and learning rate  $\eta$ , if the “+” and “-” classes are linearly separable, this algorithm is guaranteed to find a version of hyperplane in finite number of steps, that separates the classes.

