Artificial Intelligence
(Linear classifier and NN as classifier)

Prof K R Chowdhary

CSE Dept., MBM University

February 13 & 17, 2025
Lecture #6

- Classification is like, to classify a newly arriving email as spam or nospam, depending on what it is actually.
- As an example, to construct an email classifier we need many emails as examples, each provided with label spam/ nospam.
- From these we construct an email classifier some how (?), that is based on some complex relation between email content and the label. This process is called *training phase* of the classifier.
- Having trained the classifier, we will be able to label any future unlabeled email as spam or nospam. This phase is called *testing phase* of the classifier.
- If only two keywords of each email are taken criteria for classification (called attributes of the emails), then we may call these attributes as (x_1, x_2). And the system is called 2D. If there are 3 keywords for this, it is $3D$ system, and if n keywords, it is nD system.

Linear Classifier: 3D

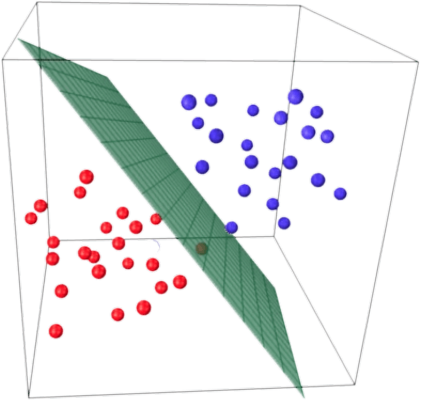

Figure 1: A linear classifier in 3D coordinates

- Similarly, examples with *n*-dimensional instance space, every example having one of two different labels, tend to cluster in two different regions.

- This observation motivates us to use an approach to classify 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 values of other attributes.
- We use a domain with attributes as real numbers, in a algebraic function (Fig.2).

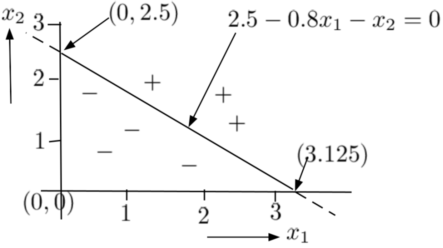

Figure 2: A linear classifier in a domain of two real valued attributes x_1, x_2

Linear Classifier

- Examples, say, $(1.0, 2.3)$, $(1.6, 1.8)$, etc. are labeled as '+' or '-', and the two classes are separated by a linear function:

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

- In equation (1)[1], variables x_1 and x_2 are real numbers.
- Problem: Given the graph in Fig. (2), construct the eq. (1).

In eqn. $y = mx + c$, constant $c = 2.5$, and $m = -2.5/3.125 = -0.8$

- Table 1 (next slide) shows seven **examples** of attributes (x_1, x_2), value of classifier "$2.5 - 0.8x_1 - x_2$," and class of each example.
- For the point (coordinate) falling below the straight line, classifier returns *neg*. When it is above, classifier returns *pos*.
- When + value is returned by the classifier, we say that example is in *pos* class. Similarly for neg value.

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

- Hence, when a classifier like this is created, we are able to classify any attribute set of two dimensional vector.
- Not only the linear classifier Eqsn. (1) classifies examples as *pos* and *neg*, but any classifier, e.g., $1.5 + 2.1x_1 - 1.1x_2 = 0$, will classify infinitely large number of examples as *pos*/*neg*. Its generic form is:

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

or even:

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

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

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:

$$\sum_{i=0}^{n} w_i x_i = 0. \qquad (4)$$

Classifier's behavior is decided by coefficients w_i (weights).

- **Task of ML**: Find out w_i's values. In equ. $y = mx + c$, m is angle w.r.t. x, y coordinate system, and in (3), coefficients $w_1, ..., w_n$ define angle of hyperplane w.r.t. system coordinates $x_1, ..., x_n$, the w_0 is *bias*/ offset: the distance of hyperplane from system coordinates.

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

- *Bias* in Fig. 2 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:

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

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

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

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

x_1	x_2	$(-0.8x_1 - x_2)$	θ
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

Perceptron Learning for binary attributes

- **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.
- Let $c(\mathbf{x})$ is "real-class", and $h(\mathbf{x})$ is "hypothesized class".
- Let for $c(\mathbf{x}) = 1$, class=*pos*, and for $c(\mathbf{x}) = 0$, it is *neg*.
- If, $\sum_{i=0}^{n} w_i x_i > 0$, classifier hypothesizes $\mathbf{x}$ as *pos* (i.e., $h(\mathbf{x}) = 1$).
- 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 there exits $h(\mathbf{x}) = c(\mathbf{x})$. Task of ML: find weights w_i that correctly classifies all $\mathbf{x}$.

Inducing the Linear Classifier

- Say classes are 0 and 1.
- *Objective*: for any attribute example $\mathbf{x}$ with "real class" $c(\mathbf{x}) = 1$, the classifier must hypothesize the example as positive, i.e. $h(\mathbf{x}) = 1$. If $c(\mathbf{x}) = 0$, it must hypothesize $\mathbf{x}$ as negative, i.e. $h(\mathbf{x}) = 0$.
- If $c(\mathbf{x}) \neq h(\mathbf{x})$, i.e., 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. Hence, weights be increased so that $\sum_{i=0}^{n} w_i x_i > 0$, (This will make, $h(\mathbf{x}) = 1$).
- It is simple to understand that only the weight of w_i be increased for which $x_i = 1$, (when $x_i = 0$, $w_i.x_i = w_i.0 = 0$).
- 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 $\sum_{i=0}^{n} w_i x_i < 0$.

Weight adjustment in Perceptron

Weight adjustment:

• When both labels same, $c(\mathbf{x}) = h(\mathbf{x})$: no weight adjustment required.

Regulate the weights by:

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

$\eta \in (0, 1]$, 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: $w_i + \eta.1.x_i = w_i + \eta$, as $x_i = 1$. This increases w_i, so it is ≥ 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: $w_i + \eta.[-1].1 = w_i - \eta$, as $x_i = 1$. This decreases w_i to ≤ 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 $c(\mathbf{x})$.

- To start with, weights w_i of perceptron are initialized to some random values. Next, each training example $\mathbf{x}$ with attributes $x_1, ..., x_i, ..., x_n$, 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 $\mathbf{x} = (x_1, ..., x_n)$, 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**

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	$c(\mathbf{x})$
Ex1	1	0	0
Ex2	1	1	1
Ex3	0	0	0

We consider that learning rate $\eta = 0.5$, and randomly generated initial weights w_0, w_1, w_2 are $[0.1, 0.3, 0.4]$ 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 4 shows.

Example on Perceptron Learning Algorithm...

Table 4: Weight adjustments for perceptron learning

Var.→ Examples ↓	x_1	x_2	w_0	w_1	w_2	$c(\mathbf{x})$	$h(\mathbf{x})$	$c(\mathbf{x})$-$h(\mathbf{x})$
Random classifier			0.1	0.3	0.4			
Ex1→	1	0				0	1	-1
New Classifier:			-0.4	-0.2	0.4			
Ex2→	1	1				1	0	1
New Classifier:			0.1	0.3	0.9			
Ex3→	0	0				0	1	-1
New Classifier:			-0.4	0.3	0.9			

The final version of classifier: $-0.4 + 0.3x_1 + 0.9x_2 = 0$ classifies correctly. After one more computation from top Ex1, we get $c(\mathbf{x}) = h(\mathbf{x}) = \mathbf{0}$.

References

[1] Chowdhary, K.R. (2020). Statistical Learning Theory. In: Fundamentals of Artificial Intelligence. Springer, New Delhi. https://doi.org/10.1007/978-81-322-3972-7_14