Unit-2
Rosenblatt's Perceptron

By: Arjun Saud, Asst. Prof. CDCSIT,TU

Introduction

- The perceptron is the simplest form of a neural network used for the classification linearly separable patterns. Patterns that lie on opposite sides of a hyperplane are called linearly separable patterns.
- Basically, perceptron consists of a single neuron with adjustable synaptic weights and bias. The algorithm used to adjust the free parameters of this neural network first appeared in a learning procedure developed by Rosenblatt (1958, 1962).

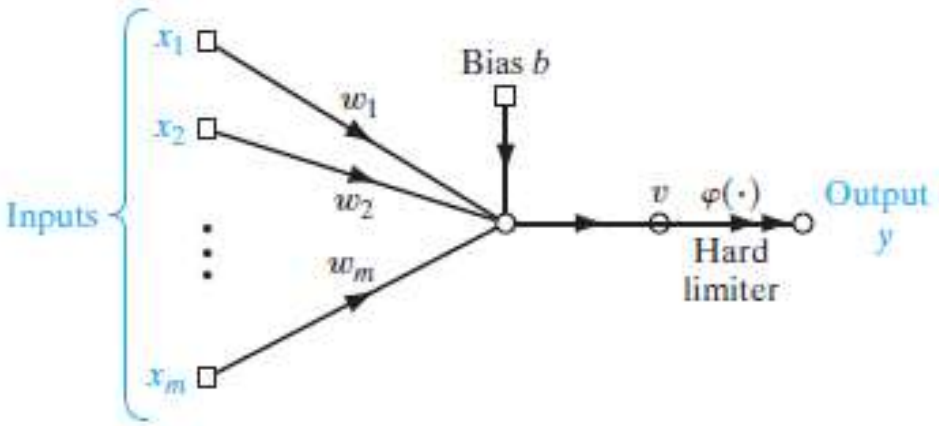
Introduction

- Rosenblatt proved that if the patterns (vectors) used to train the perceptron are drawn from two linearly separable classes, then the perceptron algorithm converges and positions the decision surface in the form of a hyperplane between the two classes.
- The perceptron built around a *single neuron* is limited to performing pattern classification with only two classes. By expanding the output layer of the perceptron to include more than one neuron, we may correspondingly perform classification with more than two classes. However, the classes have to be linearly separable for the perceptron to work properly.

Perceptron

- Rosenblatt's perceptron is built around the *McCulloch–Pitts model* of a neuron.
- The summing node of the neural model computes a linear combination of the input. The resulting sum is applied to a hard limit activation function.
- The neuron produces an output equal to 1 if the hard limiter input is positive, and -1 if it is negative.

Perceptron



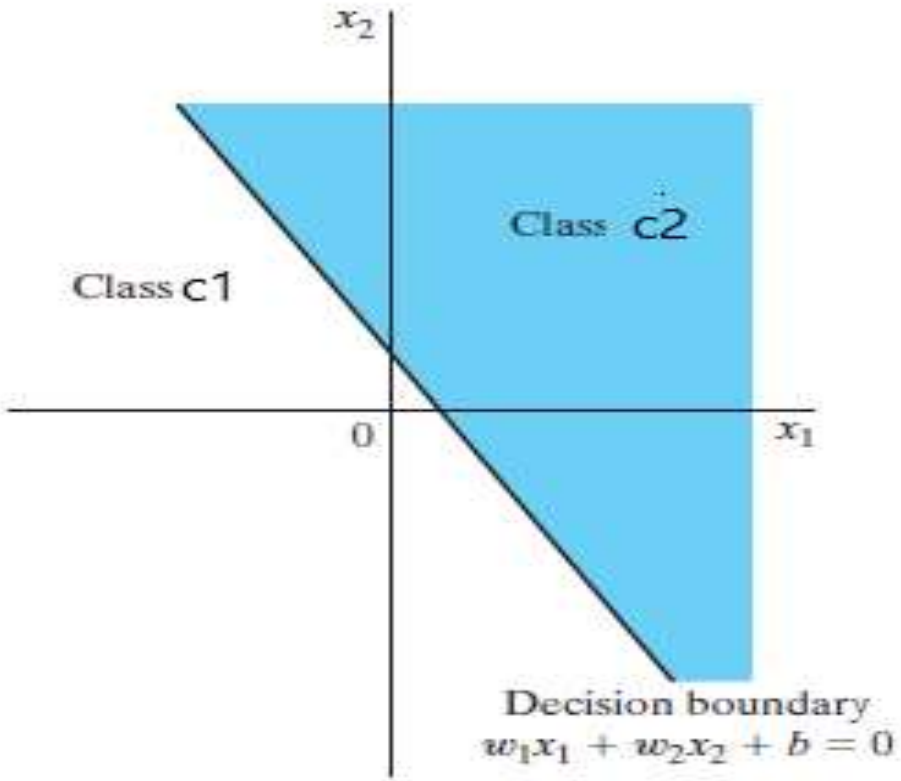
The hard limiter input (or induced local field) of the neuron is.

$$v = \sum_{i=1}^{m} w_i x_i + b$$

Perceptron

- The goal of the perceptron is to correctly classify the set of externally applied stimuli $x1$, $x2$, ..., xm into one of two classes,c1 or c2.The decision rule for the classification is to assign the point represented by the inputs $x1$, $x2$, ..., xm to class c1 if the perceptron output y is +1 and to class c2 if it is -1.
- In the simplest form of the perceptron, there are two decision regions separated by a *hyperplane*, which is defined by.

Perceptron



Perceptron

- Perceptron can be trained to show behavior or AND function and OR function because both are linearly separable. But, perceptron can not be trained to learn behavior of XOR function because it is not linearly separable.

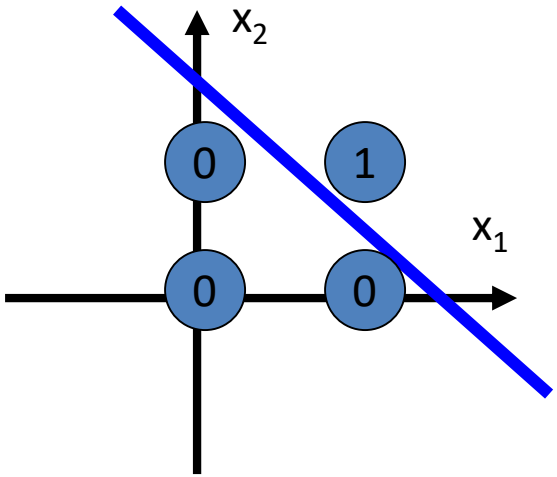


Fig: AND Function

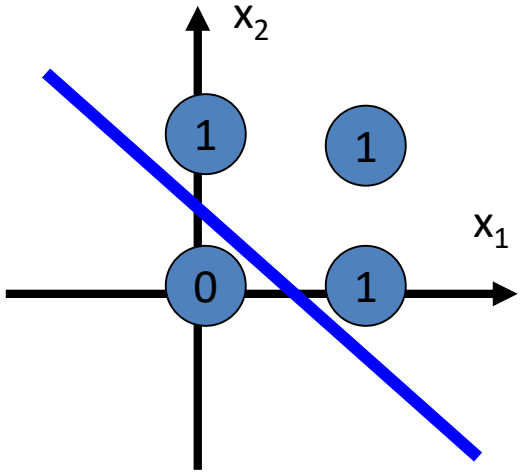


Fig: OR Function

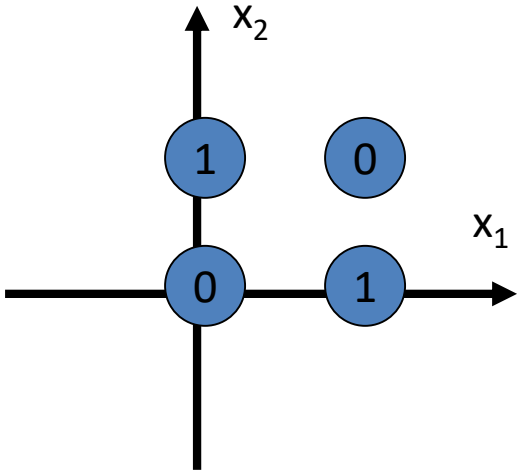


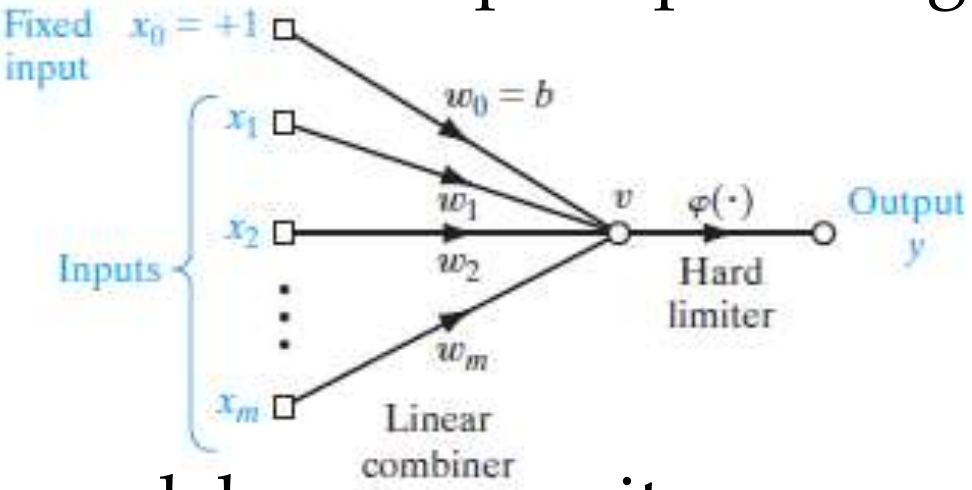
Fig: XOR Function

Perceptron Convergence Theorem

- The Perceptron convergence theorem states that *"for any data set which is linearly separable the Perceptron learning rule is guaranteed to find a solution in a finite number of steps, provided that learning rate is small."*
- Perceptron learning rule is guaranteed to converge to a weight vector that correctly classifies the examples provided the training examples are linearly separable.
- Verification of this convergence theorem is provided in the example of perceptron learning rule.

Perceptron Learning Rule

- We can derive error correction rule for perceptron. Model of perceptron is given below,



- For this model, we can write

$$v = \sum_{i=0}^{m} w_i x_i$$

- Let x=[x_0, x_1, x_2,..... x_m] and w=[w_0, w_1,..... w_m]

Perceptron Learning Rule

- Now, above equation can be written as:

$$v = w^T x$$

- Let us consider for n^{th} step input vector and weight vector is:

x=[x_0(n),x_1(n),...x_m(n)] and w=[w_0(n),w_1(n),...w_m(n)]

- Now, for n^{th} step, above equation becomes

$$v = w(n)^T x(n)$$

- Suppose and C_1 and C_2 are two classes for which perceptron needs to be trained. So, training set must include sufficient examples from both classes.

Perceptron Learning Rule

- Let H_1 and H_2 are set of training examples such that $H_1 \in C_1$ and $H_2 \in C_2$. Thus, Training Set(H)= $H_1 \cup H_2$.
- We know that perceptron puts data point in class C_1 if output is positive and in class C_2 if output is negative.
- This means, data point lies in class C_1 if v>0 and in class C_2 if v≤0. Thus,
- For each $x \in H_1$, $w^T x > 0$ and
- For each $x \in H_2$, $w^T x \leq 0$

Perceptron Learning Rule

- Now, we can write weight update rule for n^{th} step as below.

Case I : Correct Classification

if $w^T(n)x(n) > 0$ and $x(n) \in C_1$ then $w(n+1) = w(n)$

if $w^T(n)x(n) \leq 0$ and $x(n) \in C_2$ then $w(n+1) = w(n)$

Case II : Missclassification

if $w^T(n)x(n) > 0$ and $x(n) \in C_2$ then $w(n+1) = w(n) - \alpha\, x(n)$

if $w^T(n)x(n) \leq 0$ and $x(n) \in C_1$ then $w(n+1) = w(n) + \alpha\, x(n)$

Perceptron Learning Rule

- The main theme of above weight change rule is:
  - Do not change weights in case of correct classification
  - Increase weights when perceptron output is smaller than actual target
  - Decrease weights when perceptron output is larger than actual target
- Thus, we can represent weight change rule using single equation as below:

$$w(n+1) = w(n) + \alpha\, x(n)(t(n) - y(n))$$

$$where,\ y(n)\ \text{is actual output},\ and\ t(n)\ \text{is target output}$$

Perceptron Learning Rule

Perceptron Learning Algorithm

1. Initialize all weights and bias to zero
2. For each training vector s and target t perform steps *3 to 6*
3. Set $x_i = s_i \quad for\ i = 1\ to\ n$
4. Compute output using Hard limiter activation function as below

$$y_{in} = b + \sum_{i=1}^{n} w_i x_i \qquad y = f(y_{in})$$

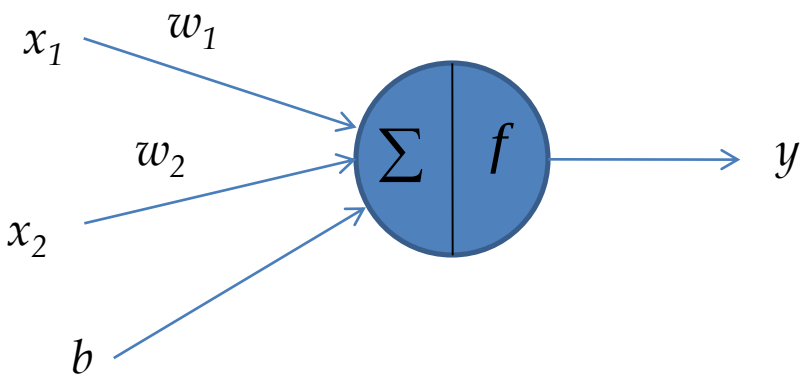
5. Adapt weights as: $w_i = w_i + \alpha(t - y)x_i \quad for\ i = 1\ to\ n$
6. Adapt bias as: $b = b + \alpha(t - y)$
7. Test for Stopping Criteria

Perceptron Learning Rule

Perceptron Training

Example

Train the following perceptron by using given training set



x_1	x_2	t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Perceptron Learning Rule

Solution: **α=1**

Epoch #1

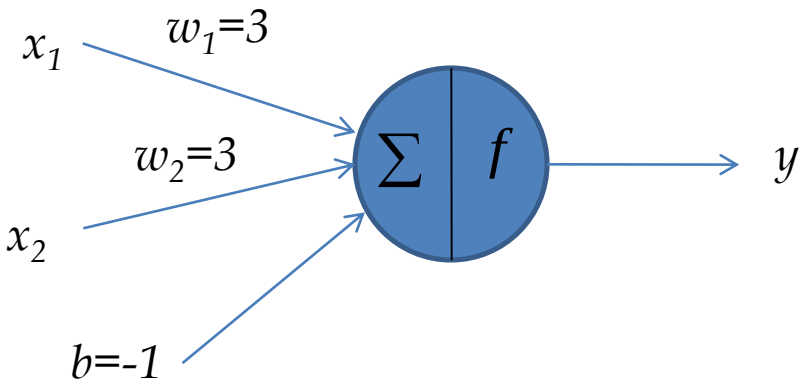
Input	w_1 *(old)*	w_2 *(old)*	*b (old)*	*v*	*y*	w_1 *(new)*	w_2*(new)*	*b(new)*
(1,1,1)	0	0	0	0	0	0+1*1*1=1	0+1*1*1=1	0+1*1=1
(1,-1,-1)	1	1	1	1	1	1+1*-2*1=-1	1+1*-2*-1=3	1+1*-2=-1
(-1,1,-1)	-1	3	-1	3	1	-1+1*-2*-1=1	3+1*-2*1=1	-1+1*-2=-3
(-1,-1,-1)	1	1	-3	-5	-1	1+1*0*-1=1	1+1*0*-1=1	-3+1*0=-3

Perceptron Learning Rule

Epoch #2

Input	w_1 *(old)*	w_2 *(old)*	*b (old)*	*v*	*y*	w_1 *(new)*	w_2*(new)*	*b(new)*
(1,1,1)	1	1	-3	-1	-1	1+1*2*1=3	1+1*2*1=3	-3+1*2=-1
(1,-1,-1)	3	3	-1	-1	-1	3+1*0*1=3	3+1*0*-1=3	-1+1*0=-1
(-1,1,-1)	3	3	-1	-1	-1	2+1*0*-1=3	3+1*0*1=3	-1+1*0=-1
(-1,-1,-1)	3	3	-1	-7	-1	3+1*0*-1=3	3+1*0*-1=3	-1+1*0=-1

Thus, Final Neuron is



Perceptron Learning Rule

Example 2

Train perceptron using given training set and predict class for the input (6,82) and (5.3,52)

Height(x_1)	*Weight(x_2)*	*Class(t)*
5.9	75	Male
5.8	86	Male
5.2	50	Female
5.4	55	Female
6.1	85	Male
5.5	62	Female

Perceptron Learning Rule

Solution

Normalize the input and encode target

Use min-max scaling y=(x-min)/(max-min)

Scaling of 5.9 *:- (5.9-5.2)/(6.1-5.2)=0.7/0.9=0.778*

Similarly normalize other values

Height(x_1)	*Weight(x_2)*	*Class(t)*
5.9	75	Male
5.8	86	Male
5.2	50	Female
5.4	55	Female
6.1	85	Male
5.5	62	Female

Height(x_1)	*Weight(x_2)*	*Class(t)*
0.778	0.694	1
0.667	1.000	1
0.000	0.000	-1
0.222	0.139	-1
1.000	0.972	1
0.333	0.333	-1

Perceptron Learning Rule

Solution: **α=1**

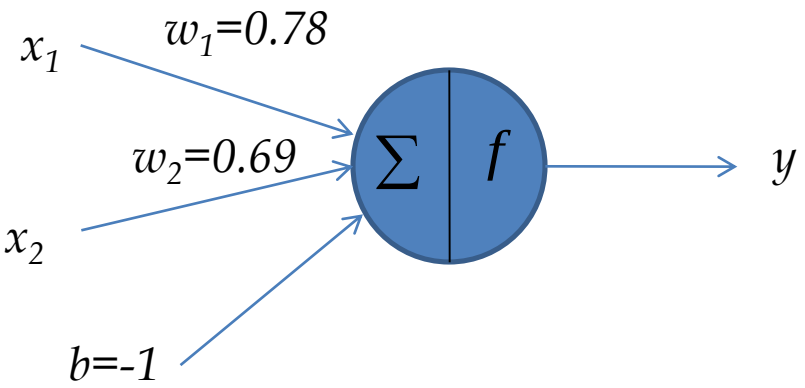
Epoch #1

Input	w_1 *(old)*	w_2 *(old)*	*b (old)*	*v*	*y*	w_1 *(new)*	w_2*(new)*	*b(new)*
0.78,0.69,1	0	0	0	0	0	0+1*1*0.78=0.78	0+1*1*0.69=0.69	0+1*1=1
0.67,1,1	0.78	0.69	1	2.21	1	0.78	0.69	1
0,0,-1	0.78	0.69	1	1	1	0.78+1*-2*0=0.78	0.69+1*-2*0=0.69	1+1*-2=-1
0.22,0.14,-1	0.78	0.69	-1	-0.73	-1	0.78	0.69	-1
1,0.97,1	0.78	0.69	-1	0.45	1	0.78	0.69	-1
0.33,0.33,1-	0.78	0.69	-1	-0.51	-1	0.78	0.69	-1

Perceptron Learning Rule

Solution Contd..

Thus, Final perceptron is as below:



For (6,82)

Normalized value of 6=(6-5.2)/(6.1-5.2)=0.8/0.9=0.89

Normalized value of 82=(82-50)/(86-50)=32/36=0.89

Perceptron Learning Rule

Solution Contd..

$v=w_1x_1+w_2x_2+b=0.78*0.89+0.69*0.89-1=0.31$

Thus, y=1

Hence, Predicted class for (6,82) is Male.

Similarly, we can predict class for (5.3,52)

Gaussian Naïve Bayes Classifier

- Normal distribution, also known as the Gaussian distribution, is a probability distribution that is symmetric about the mean, showing that data near the mean are more frequent in occurrence than data far from the mean.
- Gaussian Naïve Bayes Classifier is used to classify the data that shows Gaussian distribution.
- Let us consider a m-dimensional input vector $\{x_1, x_2,\ldots,x_m\}$ and n classes $C_1, C_2,\ldots,C_n$.
- Use training data to calculate mean and variance as below

Gaussian Naïve Bayes Classifier

$$\mu_{x_i,c_j} = \frac{1}{\mathrm{N}} \sum_{\mathrm{x_i} \in \mathrm{c_j}} x_i \qquad \text{and}$$

$$\sigma^2_{x_i,c_j} = \frac{1}{\mathrm{N}} \sum_{\mathrm{x_i} \in \mathrm{c_j}} (x_i - \mu_{x_i,c_j})^2 \qquad \text{for i} = 1 \text{ to m and j} = 1 \text{ to n}$$

where, N is the number of members is class $\mathrm{C_j}$

- Then use probability distribution function to calculate probabilities for test data as below.

$$p(x_i \mid c_j) = \frac{1}{\sigma_{x_i,c_j}\sqrt{2\pi}} \exp - \frac{1}{2}\left(\frac{\left(x_i - \mu_{x_i,c_j}\right)^2}{\sigma^2_{x_i,c_j}} \right)$$

Gaussian Naïve Bayes Classifier

$$p(x \mid c_j) = p(x_1 \mid c_j) \times p(x_2 \mid c_j) \times ..... \times p(x_m \mid c_j)$$

$$p(c_j \mid x) = \frac{p(x \mid c_j) \times p(c_j)}{p(x \mid c_1) \times p(c_1) + p(x \mid c_2) \times p(c_2) + ... + p(x \mid c_n) \times p(c_n)}$$

Since denominator is same for all $p(c_j \mid x)$ we can ignore it.

Finally, x is assigned to the class with largest value of $p(c_j \mid x)$

Gaussian Naïve Bayes Classifier

Example

Train Gaussian Bays Classifier using given training set and predict class for the input (4.5,40) and (5.8,75)

Height(x_1)	*Weight(x_2)*	*Class(t)*
5.6	72	Adult
5.8	78	Adult
4.9	45	Children
4.6	35	Children
5.7	65	Adult
5.4	58	Adult
4.7	38	Children
4.4	35	Children

Gaussian Naïve Bayes Classifier

Solution

Assume A=Adult and C=Children are two classes

$p(A) = 4/8 = 0.5 \quad \text{and} \quad p(C) = 4/8 = 0.5$

$$\mu_{x_1,A} = \frac{1}{4}(5.6 + 5.8 + 5.7 + 5.4) = 5.625$$

$$\mu_{x_1,C} = \frac{1}{4}(4.9 + 4.6 + 4.7 + 4.4) = 4.65$$

$$\mu_{x_2,A} = \frac{1}{4}(72 + 78 + 65 + 58) = 68.25$$

$$\mu_{x_2,C} = \frac{1}{4}(45 + 35 + 38 + 35) = 38.25$$

Gaussian Naïve Bayes Classifier

Solution

$$\sigma^2_{x_1,A} = \frac{1}{4}\left((5.6-5.625)^2 + (5.8-5.625)^2 + (5.7-5.625)^2 + (5.4-5.625)^2\right) = 0.0219$$

$$\sigma^2_{x_1,C} = \frac{1}{4}\left((4.9-4.65)^2 + (4.6-4.65)^2 + (4.7-4.65)^2 + (4.4-4.65)^2\right) = 0.0325$$

$$\sigma^2_{x_2,A} = \frac{1}{4}\left((72-68.25)^2 + (78-68.25)^2 + (65-68.25)^2 + (58-68.25)^2\right) = 56.188$$

$$\sigma^2_{x_2,C} = \frac{1}{4}\left((45-38.25)^2 + (35-38.25)^2 + (38-38.25)^2 + (35-38.25)^2\right) = 16.688$$

Gaussian Naïve Bayes Classifier

Solution

Now, For the data (4.5,40)

$$p(x_1 \mid A) = \frac{1}{\sqrt{2\pi\sigma_{x_1,A}^2}} \exp - \frac{1}{2}\left(\frac{\left(x_1 - \mu_{x_1,A}\right)^2}{\sigma_{x_1,A}^2} \right)$$

$$= \frac{1}{\sqrt{2\pi * 0.0219}} \exp - \frac{1}{2}\left(\frac{\left(4.5 - 5.625\right)^2}{0.0219} \right) = 0.00000000000076$$

$$p(x_2 \mid A) = \frac{1}{\sqrt{2\pi\sigma_{x_2,A}^2}} \exp - \frac{1}{2}\left(\frac{\left(x_2 - \mu_{x_2,A}\right)^2}{\sigma_{x_2,A}^2} \right)$$

$$= \frac{1}{\sqrt{2\pi * 56.188}} \exp - \frac{1}{2}\left(\frac{\left(40 - 68.25\right)^2}{56.188} \right) = 0.000044$$

Gaussian Naïve Bayes Classifier

Solution

Now, For the data (4.5,40)

$$p(x_1 \mid C) = \frac{1}{\sqrt{2\pi\sigma_{x_1,C}^2}} \exp - \frac{1}{2}\left(\frac{\left(x_1 - \mu_{x_1,C}\right)^2}{\sigma_{x_1,C}^2} \right)$$

$$= \frac{1}{\sqrt{2\pi * 0.0326}} \exp - \frac{1}{2}\left(\frac{\left(4.5 - 4.65\right)^2}{0.0326} \right) = 1.566$$

$$p(x_2 \mid C) = \frac{1}{\sqrt{2\pi\sigma_{x_2,C}^2}} \exp - \frac{1}{2}\left(\frac{\left(x_2 - \mu_{x_2,C}\right)^2}{\sigma_{x_2,C}^2} \right)$$

$$= \frac{1}{\sqrt{2\pi * 16.688}} \exp - \frac{1}{2}\left(\frac{\left(40 - 38.25\right)^2}{16.688} \right) = 0.09$$

Gaussian Naïve Bayes Classifier

Solution

$$p(x \mid A) = p(x_1 \mid A) \times p(x_2 \mid A) = 0.000000000000076 \times 0.000044$$
$$= 0.0000000000000000033$$

$$p(x \mid C) = p(x_1 \mid C) \times p(x_2 \mid C) = 1.566 \times 0.09$$
$$= 0.141$$

$$p(A \mid x) = p(x \mid A) \times p(A) = 0.0000000000000000033 * 0.5 \approx 0$$
$$p(C \mid x) = p(x \mid C) \times p(C) = 0.141 * 0.5 \approx 0.071$$

Since $p(C \mid x) > p(A \mid x)$ predicted class for (4.5,40) is C (i.e. Children)

Relationship between Perceptron and Gaussian Bayes Classifier

- For, two class Gaussian classifier it has been proved that

$$\log \Lambda(x) = (\mu_1 - \mu_2)^T C^{-1} x + \frac{1}{2}(\mu_2{}^T C^{-1} \mu_2 - \mu_1{}^T C^{-1} \mu_1)$$

Where,

$$\Lambda(x) = f_x(x|C_1) / f_x(x|C_2)$$

μ_i is mean for class C_i

C is covariance matrix

$f_x(x|C_i)$ is conditiona l PDF of random vector X given that x is drawn from C_1

- This equation is analogous to equation of perceptron. Where,

Relationship between Perceptron and Gaussian Bayes Classifier

$$y = \log \Lambda(x)$$

$$w = (\mu_1 - \mu_2)^T C^{-1}$$

$$b = \frac{1}{2}(\mu_2{}^T C^{-1} \mu_2 - \mu_1{}^T C^{-1} \mu_1)$$

Thus, we can say that when the environment is Gaussian, the Bayes classifier reduces to a linear classifier and takes same form taken by the perceptron.

Batch Perceptron Algorithm

- When training set is large, taking each training example one by one and updating perceptron weights is time consuming tasks.
- In such case batch perceptron algorithm is suitable. The main concept behind batch perceptron algorithm is: "Process training examples in batches. Size of batch should be defined on the basis of size of training set. Commonly used batch size are 16, 32, 64,128, etc. "
- Then, update weights for $(n+1)^{th}$ step by taking sum of all misclassifications in the batch of training example.

Batch Perceptron Algorithm

- Let us use following cost function for the perceptron:

$$J(w) = \sum_{x_i \in m} -w^T x_i t_i$$

where m is the set of miscalssif ied training examples

- Differentiate above cost function w.r.t w to get gradient vector.

$$\nabla J(w) = \sum_{x_i \in m} -x_i t_i$$

- In the *steepest descent method, the adjustment to the weight vector* ***w at each time step*** of the algorithm is applied in a direction *opposite to the gradient vector. Thus,*

Batch Perceptron Algorithm

- Thus, weight update rule for batch perceptron algorithm can be written as

$$w(n+1) = w(n) - \alpha \nabla J(w) = w(n) + \alpha \sum_{x_i \in m} x_i t_i$$