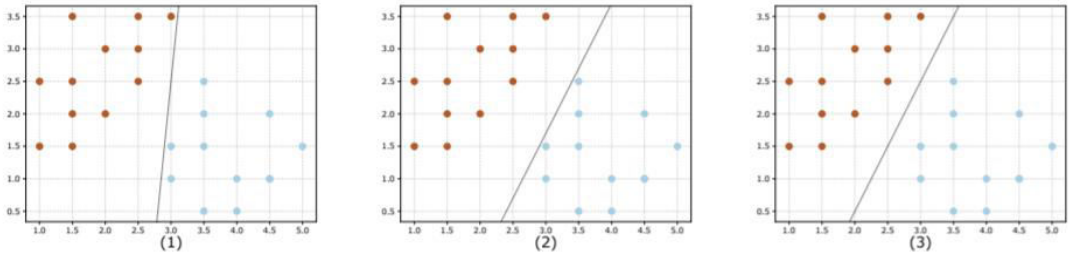
Anna Krasilnikova 2562668
Mirza Misbah Mubeen Baig 2571567
Shahzain Mehboob 2571564

Exercise 7.1 – *Support Vector Machines*

a) *Given the following 2D dataset with data from two classes and we want to train a linear classifier to separate samples belonging to different classes. Which of the following classifiers is the most suitable for the problem?*



The most suitable classifier is shown on the figure 3. It shows the hyperplane, correctly classifying the data and having the maximum margin to the data.

Most likely, the new data would be classified correctly if there exists the distance between separation hyperplane and marginal data instead of just having the hyperplane in the near of those data points. In this case the new data can be top close to the marginal point whi ch would lead to the wrong classification.

b) *A simple binary Support Vector Machine classifier can be defined by:*

$$\hat{y}_i = sgn(w^T x^{(i)} + b)$$

with $sgn(z) = \begin{cases} 1, & if\ z < 0 \\ -1, & if\ z < 0 \end{cases}$

with the prediction $\hat{y}_i \in \{-1, 1\}$ *of the data sample* $x^{(i)} \in \mathbb{R}^n$ *(here n=2) given the trained weights w and the bias b.*

Using a subset of the visible data points from a) as training set X, a linear SVM model learned the following values for α and b (see chapter 5, slide 35):

$$X = \begin{bmatrix} 2.5 & 2 & 2.5 & 1.5 & 1.5 & 2.5 & 2 & 3 & 3.5 & 3.5 & 3.5 & 4 & 3 & 4 & 3.5 & 3 \\ 3.5 & 3 & 3 & 1.5 & 2 & 2.5 & 2 & 3.5 & 2 & 1.5 & 0.5 & 1 & 1.5 & 0.5 & 2.5 & 1 \end{bmatrix}^T$$

$$\alpha = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 2.48 \quad 0 \quad 0.02 \quad 0 \quad 0 \quad 0 \quad 0 \quad -0.98 \quad 0 \quad -1.52 \quad 0\ ]^T$$

And $b = 3.5$

Use this SVM to classify the points $p_1 = (4, 1.5)^T$ *and* $p_2 = (3, 2.5)^T$*. State each of your computational steps explicitly and solve this exercise without an SVM framework implementation.*

So, let's start from representing this classifier, taking into the consideration chapter 5, slide 35, as following:

$$w^T x + b = b + \sum_{i=1}^{m} \alpha_i \underbrace{x^T x^{(i)}}_{\text{the innter product of ith training sample with respect to } x}$$

We have non-zero values with indices 6, 8, 13,15.

Thus, let's represent the output for the classifier as:

$\sum_{i=1}^{m} \alpha_i x^T x^{(i)} + 3.5$, where $i \in \{6, 8, 13, 15\}$

Let's consider 2 points:

$$p_1 = (4, 1.5)^T \text{ and}$$

$$p_2 = (3, 2.5)^T$$

For the point $p_2 = (4, 1.5)^T$ we have following:

$\hat{y}_1 = 3.5 + [(4, 1.5)(2.5, 2.5)^T 2.48 + (4, 2.5)(3.5, 3.0)^T 0.02 + (4, 1.5)(1.5, 3.0)^T(-0.98) + (4, 1.5)(2.5, 3.5)^T(-1.52)] = 4.44$, which makes the point p_1 class 1, since it's $\hat{y}_1 \geq 0$.

Anna Krasilnikova 2562668
Mirza Misbah Mubeen Baig 2571567
Shahzain Mehboob 2571564

Thus, we can calculate the point $p_2 = (3,2.5)^T$:

$\hat{y}_2 = 3.5 + [(3,2.5)(2.5,2.5)^T 2.48 + (3,2.5)(3.5,3.0)^T 0.02 + (3,2.5)(1.5,3.0)^T(-0.98) + (3,2.5)(2.5,3.5)^T(-1.52)] = 1.5$, which makes the point p_2 class 1, $\hat{y}_2 \geq 0$

c) *Support Vector machines are also called "Maximum Margin Classifiers". Explain this name as well as the term "support vector" w.r.t. the values of α.*

We know, that by definition, using SVNs we obtain a separating hyperplane, which essentially mean two things:

- They are laying closest to decision surface and
- They are quite hard to classify.

Support vectors are basically the subset of training samples, specifying the decision function.

Now we can define a separation distance as the separation between the closest to the hyperplane data point for a certain vector of weight w and bias b.

Optimal hyperplane determines the hyperplane with maximized margin of separation d.

In this case, α has non-zero values, corresponding to support vectors, which are the Lagrangian multipliers, used for solving the optimization problem instead of weights.

d) *What can be done with SVMs when data samples are not linear separable? Briefly explain what your approach does.*

We can start by defining D as the basis functions/feature which maps φ_i as φ_i: $\mathbb{R}^d \rightarrow \mathbb{R}$:

And we can define $x \in \mathbb{R}^d \rightarrow (\varphi_1(x), \ldots, \varphi_D(x))$, which maps the original input space

$x \in \mathbb{R}^D$ to a possibly larger feature space $\mathbb{R}^D$.

Given that, we can define following:

$$f(x) = sign\,(\langle w, \phi(x)\rangle + b) = sign \sum_{i=1}^{D} (\langle w, \phi(x)\rangle + b)$$

Since we know that functions with linear parameters are not necessarily linear in the input space, this equation can be rewritten by substituting the given inner $x^T x^{(i)}$ product to the new inner product $k(x, x^{(i)})$.

Then we'll get following function:

$$b + \sum_{i=1}^{m} \alpha_i k(x, x^{(i)})$$

This allows us to model highly non-linear decision boundaries, since the problem from non-linearly separable is converting to the linearly separable one.

This allows to modify highly non-linear decision boundaries: this way we can transform into a problem which is linearly separable from the problem which is non-linearly separable.

Exercise 7.2 – *Validation and Cross-Validation*

Forward propagation first pass,

Input layer:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Anna Krasilnikova 2562668
Mirza Misbah Mubeen Baig 2571567
Shahzain Mehboob 2571564

$$w_{hidden} \Longrightarrow \begin{bmatrix} w_1^1 & w_2^1 & w_3^1 \\ w_1^2 & w_2^2 & w_3^2 \end{bmatrix} = \begin{bmatrix} 0.15 & -0.25 & 0.05 \\ 0.2 & 0.1 & -0.15 \end{bmatrix}$$

$a = w_{hidden}^T x \Longrightarrow h = \sigma(a)$, where $\sigma = \frac{1}{1+e^{-a}}$

Let's calculate a:

$$a = w_{hidden}^T x = \begin{bmatrix} 0.15 & 0.2 \\ -0.25 & 0.1 \\ 0.05 & -0.15 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.51 \\ 0.58 \\ 0.45 \end{bmatrix}$$

Now finding $\hat{y}$, we know that:

$$\hat{y} = \sigma(a^2)$$

where $a^2 = w^r h^1$

We first calculate a^2.

$$a^2 = [0.2 \quad -0.35 \quad 0.15] \begin{bmatrix} 0.51 \\ 0.58 \\ 0.45 \end{bmatrix} = -0.09$$

Now

$$\hat{y} = \sigma(a^2) = \frac{1}{1 + e^{-a^2}} = \frac{1}{1 + e^{0.09}} = 0.47$$

$$\hat{y} = 0.47$$

Now for calculating the loss, we know that the true label is $\hat{y} = 1$. Therefore, for the given input, we have:

$$L = \frac{1}{n} \sum_{i=1}^{n} -y_i \log(\hat{y}_i) - (1 - y_i) \log(1 - \hat{y}_i) = -1(-0.32) - 0 = 0.32$$

$$L = 0.32$$

Back propagation:

For back propagation we will compute the derivative of the error w.r.t. each weight connecting to hidden units to the output unit using chain rule:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a^2} \frac{\partial a^2}{\partial w}$$

Examining each factor in turn:

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} = \frac{0.47 - 1}{0.47(1 - 0.47)} = -\frac{0.53}{0.25} = -2.12$$

$$\frac{\partial \hat{y}}{\partial a^2} = \hat{y}(1 - \hat{y}) = 0.25$$

$$\frac{\partial a^2}{\partial w} = h = \begin{bmatrix} 0.51 \\ -0.58 \\ 0.45 \end{bmatrix}$$

Combining these terms together we get:

$$\delta w = \begin{bmatrix} -2.12 \times 0.25 \times 0.51 \\ -2.12 \times 0.25 \times 0.58 \\ -2.12 \times 0.25 \times 0.45 \end{bmatrix}$$

$$\delta w = \begin{bmatrix} -0.27 \\ -0.23 \\ -0.23 \end{bmatrix}$$

Considering a learning rate of 0.1, we get our find weight vector for the output layer as:

$$\delta w = \begin{bmatrix} -2.12 \times 0.25 \times 0.51 \\ -2.12 \times 0.25 \times 0.58 \\ -2.12 \times 0.25 \times 0.45 \end{bmatrix}$$

Anna Krasilnikova 2562668
Mirza Misbah Mubeen Baig 2571567
Shahzain Mehboob 2571564

$$w = \begin{bmatrix} w_1 - lr \times \delta w_1 \\ w_2 - lr \times \delta w_2 \\ w_3 - lr \times \delta w_3 \end{bmatrix}$$

$$w = \begin{bmatrix} 0.2 - 0.1 \times (-0.27) \\ -0.35 - 0.1 \times (-0.30) \\ 0.15 - 0.1 \times (-0.23) \end{bmatrix}$$

$$w_{out} = \begin{bmatrix} -0.27 \\ -0.30 \\ -0.23 \end{bmatrix}$$

The above weight vector gives us the weight w.r.t. last layer.

Now computing the gradients w.r.t. weights connecting the inputs to the hidden layer unit requires another application of the chain rule.

Here it's useful to calculate the quantity $\frac{\partial L}{\partial a'}$, where a' is weighted input sum and $h = \frac{1}{1+e^{-a'}}$ is the activation unit.

$$\frac{\partial L}{\partial a'} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial a'}$$

$$= (\hat{y} - y)(w')(h(1h))$$

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial a}\frac{\partial a}{\partial x} = \frac{\partial L}{\partial \hat{y}}\hat{y}(1-\hat{y})w$$

Then a weight w^0_{hidden} (connecting input unit to hidden unit) has gradient:

$$\frac{\partial L}{\partial w_{hidden}} = \frac{\partial L}{\partial a'}\frac{\partial a'}{\partial w_{hidden}} = (\hat{y} - y)\,(w')\,\big(h(1-h)\big)\,(x)$$

$$= -0.53 \times \begin{bmatrix} 0.15 & 0.2 \\ -0.25 & 0.1 \\ -0.05 & -0.15 \end{bmatrix}$$

$$\frac{\partial h}{\partial a} = \frac{\partial Signoid(a)}{\partial a} = h(1-h)\begin{bmatrix} 0.51(1-0.51) \\ 0.58(1-0.58) \\ 0.45(1-0.45) \end{bmatrix}$$

$$\frac{\partial h}{\partial a} = \begin{bmatrix} 0.25 \\ 0.24 \\ 0.25 \end{bmatrix}$$

$$\frac{\partial a}{\partial w} = x$$

$$\frac{\partial L}{\partial w_{hidden}} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial a}\frac{\partial a}{\partial w}$$

$$\frac{\partial L}{\partial w_{hidden}} = \frac{\partial L}{\partial \hat{y}}\frac{\partial y'}{\partial a}\frac{\partial a}{x}$$

$$\frac{\partial L}{\partial w_{hidden}} = \begin{bmatrix} 0.4 & 0.2 & 0.27 \\ 0.3 & 0.5 & -0.5 \end{bmatrix}$$

$$w_{hidden} = w_{hidden} - (epsilon)\frac{\partial L}{\partial w_{hidden}}$$

$$w_{hidden} = \begin{bmatrix} 0.15 & -0.25 & 0.05 \\ 0.2 & 0.1 & -0.15 \end{bmatrix} - 0.1 * \begin{bmatrix} 0.4 & 0.2 & 0.27 \\ 0.3 & 0.5 & -0.5 \end{bmatrix}$$

$$w_{hidden} = \begin{bmatrix} 0.11 & -0.27 & 0.023 \\ 0.17 & 0.05 & -0.1 \end{bmatrix}$$

Anna Krasilnikova 2562668
Mirza Misbah Mubeen Baig 2571567
Shahzain Mehboob 2571564

Forward propagation again,

Input layer: The input layer consists of the same two nodes x_1 and x_2 as we have previously.

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Hidden layer: The updated hidden layer is made up of 3 neurons. The corresponding matrix of weights is given as:

$$w_{hidden} \Longrightarrow \begin{bmatrix} w_1^1 & w_2^1 & w_3^1 \\ w_1^2 & w_2^2 & w_3^2 \end{bmatrix} = \begin{bmatrix} 0.11 & -0.27 & 0.023 \\ 0.17 & 0.05 & -0.1 \end{bmatrix}$$

Output layer: The updated output layer consists of one neuron, i.e., the network generates a single output. The weight matrix corresponding to the Output layer is given by: hidden layer is made up of 3 neurons. The corresponding matrix of weights is given as:

$$w_{out} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} -0.27 \\ -0.30 \\ -0.24 \end{bmatrix}$$

We know that, $a = w_{hidden}^T x \Longrightarrow h = \sigma(a)$, where $\sigma = \frac{1}{1+e^{-a}}$

Let's calculate a:

$$a = w_{hidden}^T x = \begin{bmatrix} 0.11 & 0.17 \\ -0.27 & 0.05 \\ 0.023 & -0.1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.51 \\ 0.57 \\ 0.46 \end{bmatrix}$$

Now finding $\hat{y}$, we know that:

$$\hat{y} = \sigma(a^2)$$

where $a^2 = w^r h^1$

We first calculate a^2.

$$a^2 = [-0.27 \quad -0.30 \quad -0.24] \begin{bmatrix} 0.51 \\ 0.57 \\ 0.46 \end{bmatrix} = -0.4167$$

Now

$$\hat{y} = \sigma(a^2) = \frac{1}{1 + e^{-a^2}} = \frac{1}{1 + e^{0.4167}} = 0.39$$

$$\hat{y} = 0.47$$

Now for calculating the loss, we know that the true label is $\hat{y} = 1$. Therefore, for the given input, we have:

$$L = \frac{1}{n} \sum_{i=1}^{n} -y_i \log(\hat{y}_i) - (1 - y_i) \log(1 - \hat{y}_i) = -1(-0.4008) - 0 = 0.4008$$

$$L = 0.40$$

The calculated Loss value tells us how the model is at making predictions for a given set of parameters.

The first forward propagation passes the whole data through the network and finds the error rate for all of them and then after updating weights by gradients with respect to all the data samples and apply second pass to updated weights after passing the whole data-set. That means for each epoch, passing the whole data-set through the network, one update occurs. This update is accurate toward descending gradient.