PS4: SVM and Neural Networks

DS 5220: Supervised Machine Learning and Learning Theory
Omer Seyfeddin Koc

April 18, 2023

1 SVM on a Toy Dataset (40%)

You are given a data set D with a single feature $x \in R$ and a corresponding label $y \in \{+, -\}$. The data set contains three positive examples at x = {-3, -2, 3} and three negative examples at x = {-1, 0, 1}

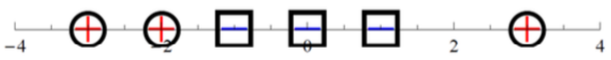

Figure 1: Data set for SVM feature map task

1. Can this data set (in its current feature space) be perfectly separated using a linear separator? Why or why not?

   Solution : The current feature space of the data set does not allow for perfect separation using a linear separator. The data changes class twice along only one dimension, making it impossible for a linear classifier in one dimension to accurately represent all the splits. Therefore, we need to explore the option of mapping the data to a different feature space in order to achieve better classification results.

2. Let's define a simple feature map $\phi(u) = \left(u, u^2\right)$ which transforms points in $\mathbb{R}$ to points in $\mathbb{R}^2$. Apply ϕ to the data and plot the points in the new $\mathbb{R}^2$ feature space.

   Solution : I used python to create the graph containing the transformation and indicating the points.

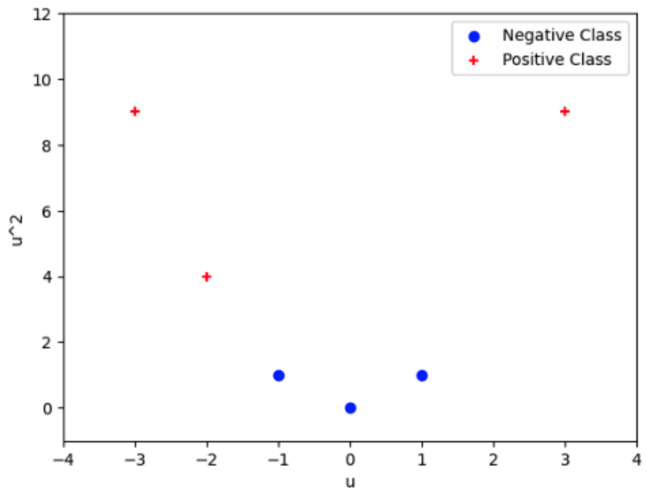

Figure 2: Scatter Plot u and u^2 (by Class)

3. Can a linear separator perfectly separate the points in the new $\mathbb{R}^2$ feature space induced by ϕ ? Why or why not?

   Solution : Yes, in the new $\mathbb{R}^2$ feature space, there are many different linear separators perfectly separate the points. An example is shown in the graph below.

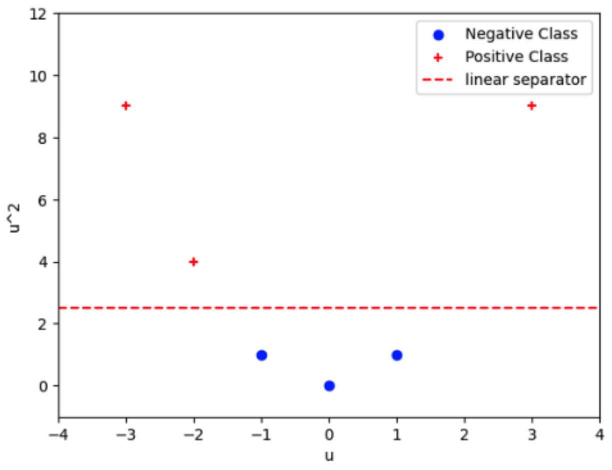

Figure 3: Linear Separator

4. Give the analytic form of the kernel that corresponds to the feature map ϕ in terms of only x and x'. Specifically define $k(x, x')$.

   Solution :

$$k(x, x') = \phi(x)^T \phi(x') = \sum_{i=1}^{m} \phi_i(x)\phi_i(x')$$

   We have $\phi_1(u) = u$ and $\phi_2(u) = u^2$

$$\begin{aligned} K(X_1, X_1') =& \sum_{j=1}^{2} \phi_j(x_1)\phi_j(x_1') \\ =& \phi_1(x_1)\phi_1(x_1') + \phi_2(x_1)\phi_2(x_1') \\ =& X_1X_1' + X_1^2(X_1')^2 \\ =& X_1X_1'(1 + X_1X_1') \end{aligned}$$

5. Construct a maximum-margin separating hyperplane. The hyperplane will be a line in $\mathbb{R}^2$, which can be parameterized by its normal equation, i.e., $w_1z_1 + w_2z_2 + c = 0$ for appropriate choices of w_1, w_2, c. Here, $(z_1, z_2) = \phi(x)$ is the result of applying the feature map ϕ to the original feature x.

   Give the values for w_1, w_2, c. Also, explicitly compute the margin for your hyperplane. You do not need to solve a quadratic program to find the maximum margin hyperplane. Instead, let your geometric intuition guide you.

   Solution : In this problem, we will use geometric intuition. The margin γ is the distance from the separating hyperplane to the closest points from both classes. A maximum-margin hyperplane must lie exactly in the middle between the classes. That's why, in the graph, the line must pass where is between (-2,4) and (-1,1) points. They are the closest points of opposite class.

   First, the point between these two points must be calculated (midpoint). Midpoint is basically the halfway between two end points. All we need to do is dividing the sum of x-values and the sum of y-values by 2. Midpoint of the points (-2,4) and (-1,1):

$$(\frac{-2 + (-1)}{2}, \frac{4 + 1}{2}) = (\frac{-3}{2}, \frac{5}{2}) = (-1.5, 2.5)$$

Also, the hyperplane should be orthogonal to the line that connects these two points. Therefore, the slope of the hyperplane will be calculated by the formula:

$$m_{hyperplane} = \frac{-1}{m} = \frac{-1}{\frac{y_2-y_1}{x_2-x_1}} = \frac{x_1 - x_2}{y_2 - y_1} = \frac{-2-(-1)}{1-4} = \frac{-1}{-3} = \frac{1}{3}$$

The formula of the line with a slope of $\frac{1}{3}$ and passing through the point (-1.5 , 2.5) is as follows:

$$\frac{1}{3}(x - (-1.5)) = (y - 2.5)$$
$$x + 1.5 = 3(y - 2.5)$$
$$x + 1.5 = 3y - 7.5$$
$$3y - x - 9 = 0$$

Substitute the z_1 and z_2 values in the formula:

$$-Z_1 + 3Z_2 - 9 = 0$$
$$\frac{-1}{3}Z_1 + Z_2 - 3 = 0$$

So, values of w_1, w_2 and c are $w_1 = \frac{-1}{3}$, w_2=1, and c=-3.

At this point, it's crucial to make sure that no other points are encroaching on the margin. There are no other points present in the margin or along it in this dataset. Additional points would require us to create a new separation hyperplane taking them into account as well. For this example, our margin is equal to half the distance between points $u_1 = (-2, 4)$ and $u_2 = (-1, 1)$.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$
$$= \sqrt{(-2 - (-1))^2 + (4 - 1)^2}$$
$$= \sqrt{10}$$

So, margin of hyperplane is $\frac{\sqrt{10}}{2}$.

6. On the plot of the transformed points (from part 2), plot the separating hyperplane and the margin, and circle the support vectors.

Solution :

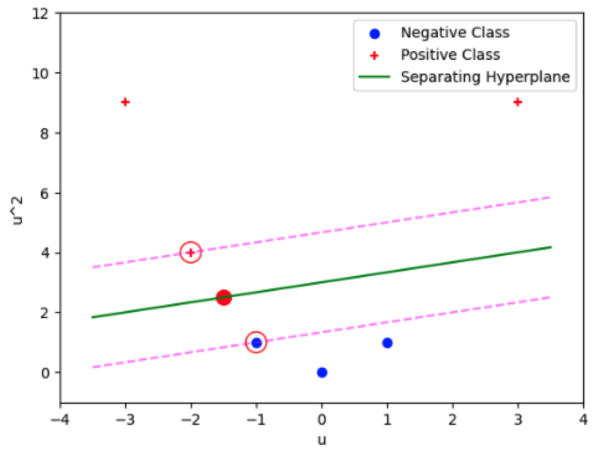

Figure 4: Separating hyperplane, the margin, and the support vectors.

7. Draw the decision boundary of the separating hyperplane in the original feature space.

   Solution :

   Decision boundary of the separating hyperplane shown in below.

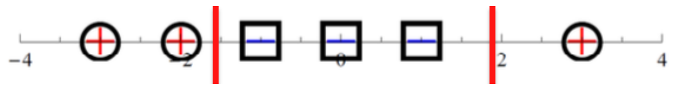

Figure 5: Decision Boundary

8. Compute the coefficients α and the constant b in Equation (1) for the kernel k and the support vectors $SV = \{u_1, u_2\}$ you found in part 6.

$$h(x) = \text{sign}\left(\sum_{i=1}^{|SV|} \alpha_i y_i k(x, u_i) + b\right)$$

   Hint: Think about the dual form of the quadratic program and the constraints placed on the α values.

   Solution : Recall that the dual form of the quadratic program.

$$\begin{aligned} \text{maximize} \quad & L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ \text{subject to} \quad & \alpha_i \geq 0 \quad \text{for } i = 1, 2, ..., N \\ & \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned}$$

   Substituting values for alphas in the $\sum_{i=1}^{N} \alpha_i y_i = 0$.

$$\begin{aligned} a_1(1) + a_2(-1) &= 0 \\ a_1 - a_2 &= 0 \\ a_1 &= a_2 \\ a_1, a_2 &= a \end{aligned}$$

   Call b and *kernel* equation:

$$b = y_i - \sum_{j=1}^{|SV|} \alpha_j y_j k(x, u_i)$$
$$K(X_1, X_1') = X_1 X_1' (1 + X_1 X_1')$$

   Now, for $u_i = -2$

$$\begin{aligned} b =& 1 - \sum_j \alpha y_j k(x_j, -2) \\ =& 1 - (\alpha k(-2, -2) - \alpha k(-1, -2)) \\ =& 1 - (\alpha(4(1+4)) - \alpha(2(1+2)) \\ =& 1 - (20\alpha - 6\alpha) \\ =& 1 - 14\alpha \end{aligned}$$

Again, for $u_i = -1$

$$\begin{aligned} b &= -1 - \sum_j \alpha y_j k\left(x_j, -1\right) \\ &= -1 - (\alpha k(-2,-1) - \alpha k(-1,-1)) \\ &= -1 - (\alpha(2(1+2)) - \alpha(1(1+1)) \\ &= -1 - (6\alpha - 2\alpha) \\ &= -1 - 4\alpha \end{aligned}$$

From above we can get α by:

$$\begin{aligned} 1 - 14\alpha &= -1 - 4\alpha \\ 2\alpha &= 10 \\ \alpha &= \frac{1}{5} \end{aligned}$$

From above we can get constant b by:

$$\begin{aligned} b &= -1 - 4(\frac{1}{5}) \\ b &= -\frac{9}{5} \end{aligned}$$

9. If we add another positive ($y = +$) point to the training set at $x = 5$, would the hyperplane or margin change? Explain.

   Solution : No. If there are additional points beyond the margin, they would not affect the position or orientation of the hyperplane. This is because the margin is specifically designed to have no points within it, and any points that lie beyond the margin do not have any influence on the hyperplane. The hyperplane will not change because the point lies beyond the margin and is classified correctly.

2 Perceptron (20%)

You are given the following training set:

Sample	x_1	x_2	x_3	y
1	1	4	1	1
2	1	2	3	1
3	0	0	1	1
4	-1	4	0	1
5	1	0	-2	0
6	-1	-1	1	0
7	0	-4	0	0
8	1	0	-3	0

(1)

Run two iterations of the perceptron learning algorithm on this data set. Start with initial weights of 0, and use a learning rate of $\alpha = 0.5$. Don't forget to add a bias neuron $x_0 = 1$.

Solution : First of all, the X (input) vector and the W (weight) vector will be multiplied for each sample (sum product), respectively. According to the O (output) value formed as a result of this sum product, the estimation (z) is made with our activation function. Then, weights are updated with the weight update rule in case of error by comparing the actual (y) value with estimation values (z). Learning rate of $\alpha = 0.5$, and bias neuron $n_0 = 1$. Also, weight update rule is:

$$w \leftarrow w + \alpha(y_i - h(x_i))x_i$$

In addition, activation function is not specified in the question. Therefore, I use the Heaviside step function as the activation function.

$$H(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

The table showing all the necessary values for the first iteration is as follows:

Sample	x_0	x_1	x_2	x_3	b	w_1	w_2	w_3	z	o	y	*error*
1	1	1	4	1	0	0	0	0	0	1	1	0
2	1	1	2	3	0	0	0	0	0	1	1	0
3	1	0	0	1	0	0	0	0	0	1	1	0
4	1	-1	4	0	0	0	0	0	0	1	1	0
5	1	1	0	-2	0	0	0	0	0	1	0	-1
6	1	-1	-1	1	-0.5	-0.5	0	1	1	1	0	-1
7	1	0	-4	0	-1	0	0.5	0.5	-3	0	0	0
8	1	1	0	-3	-1	0	0.5	0.5	-2.5	0	0	0

(2)

Weight vector resulting from the first iteration is: $w = (-1, 0, 0.5, 0.5)$. Using these weight values, we will re-estimate and update weight for each sample. This will be the second iteration. Second iteration table is as follows:

Sample	x_0	x_1	x_2	x_3	b	w_1	w_2	w_3	z	o	y	*error*
1	1	1	4	1	-1	0	0.5	0.5	1.5	1	1	0
2	1	1	2	3	-1	0	0.5	0.5	1.5	1	1	0
3	1	0	0	1	-1	0	0.5	0.5	-0.5	0	1	1
4	1	-1	4	0	-0.5	0	0.5	1	1.5	1	1	0
5	1	1	0	-2	-0.5	0	0.5	1	-2.5	0	0	0
6	1	-1	-1	1	-0.5	0	0.5	1	0	1	0	-1
7	1	0	-4	0	-1	0.5	1	0.5	-5	0	0	0
8	1	1	0	-3	-1	0.5	1	0.5	-2	0	0	0

(3)

After two iteration we get the correct weights: $b = -1, w_1 = 0.5, w_2 = 1, w_3 = 0.5$.

3 MNIST Image Classification (40%)

In this problem, you will implement a simple neural network to classify grayscale images of handwritten digits (0-9) from the MNIST data set. The data set contains 60,000 training images and 10,000 testing images of handwritten digits. Each image is 28 × 28 pixels in size, and is generally represented as a flat vector of 784 numbers. It also includes labels for each example, a number indicating the actual digit (0-9) handwritten in that image. A sample of a few such images are shown below.

You can download the starter code for this task from Canvas (*nn_starter.py*). The starter code splits the set of 60,000 training images and labels into a sets of 50,000 examples as the training set and 10,000 examples for validation set. You will implement a neural network with a single hidden layer and cross-entropy loss, and train it with the provided data set. Use the sigmoid function as activation for the hidden layer, and softmax function for the output layer.

Recall that for a single example $(\mathbf{x}, y)$, the cross-entropy loss is:

$$CE(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$

where $\hat{y} \in \mathbb{R}^K$ is the vector of softmax outputs from the model for the training example $\mathbf{x}$, and $y \in \mathbb{R}^K$ is the ground-truth vector for the training example $\mathbf{x}$ such that $y = [0, \ldots, 0, 1, 0, \ldots, 0]^T$ contains a single 1 at the position of the correct class (also called a "one-hot" representation).

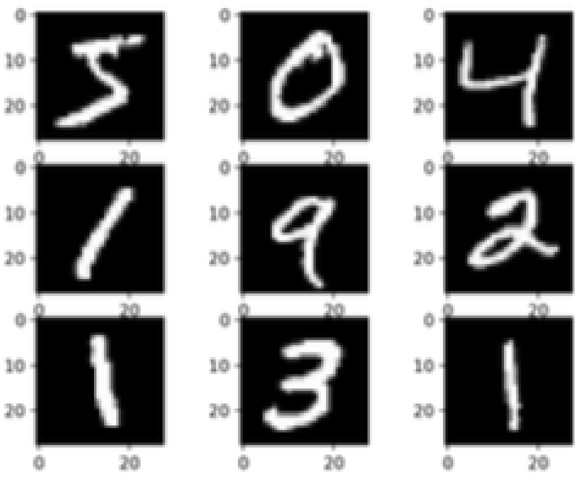

Figure 6: Numbers

The starter code already converts labels into one-hot representations for you. For n training examples, we average the cross-entropy loss over the n examples:

$$J\left(W^{[1]}, W^{[2]}, b^{[1]}, b^{[2]}\right) = \frac{1}{n}\sum_{i=1}^{n} CE\left(y^{(i)}, \hat{y}^{(i)}\right) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_k^{(i)} \log \hat{y}_k^{(i)}$$

Instead of batch gradient descent or stochastic gradient descent, common practice is to use mini-batch gradient descent for deep learning tasks. In this case, the cost function is defined as follows:

$$J_{MB} = \frac{1}{B} CE\left(y^{(i)}, \hat{y}^{(i)}\right)$$

where B is the batch size, i.e., the number of training examples in each mini-batch.

1. Implement both forward-propagation and back-propagation for the above loss function. Initialize the weights of the network by sampling values from a standard normal distribution. Initialize the bias/intercept term to 0 . Set the number of hidden units to be 300 , and learning rate to be 5 . Set $B = 1,000$ (mini batch size). This means that we train with 1,000 examples in each iteration. Therefore, for each epoch, we need 50 iterations to cover the entire training data.

   The images are pre-shuffled. So you don't need to randomly sample the data, and can just create mini-batches sequentially.

   Solution : The answers to the related question are shown in the OmerKoc_PS4_Q3.ipynb.

2. Train the model with mini-batch gradient descent as described above. Run the training for 30 epochs. At the end of each epoch, calculate the value of loss function averaged over the entire training set, and plot it (y-axis) against the number of epochs (x-axis). In the same figure, plot the value of the loss function averaged over the validation set, and plot it against the number of epochs.

   Solution : At the end of each epoch, Loss and Accuracy values were calculated for the Train and Validation sets. The related code and outputs in the OmerKoc_PS4_Q3.ipynb.

3. Similarly, in a new figure, plot the accuracy (on y-axis) over the training set, measured as the fraction of correctly classified examples, versus the number of epochs (x-axis). In the same figure, also plot the accuracy over the validation set versus number of epochs. Submit both plots along with the code.

   Solution : The related code and graphs in the OmerKoc_PS4_Q3.ipynb file.

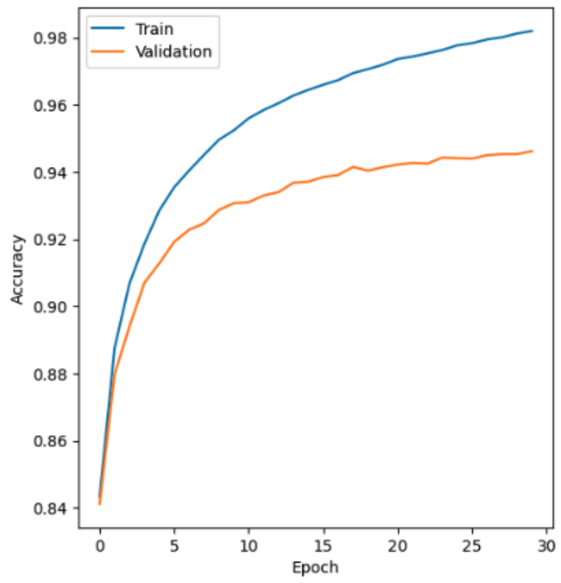

Figure 7: Accuracy-Epoch Graph

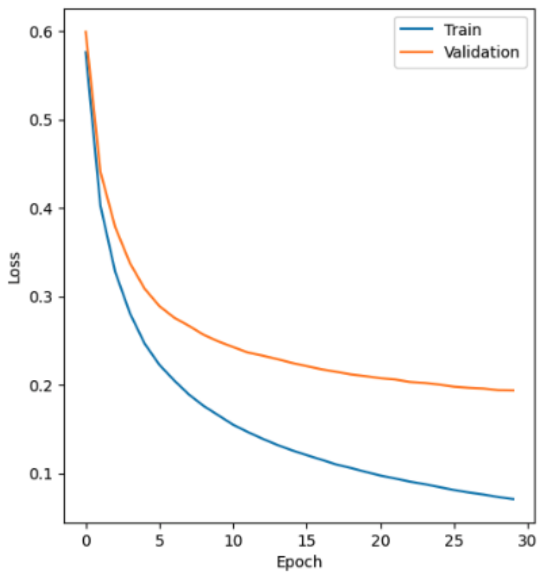

Figure 8: Loss-Epoch Graph

4. All this while you should have stayed away from the test data completely. Now that you have convinced yourself that the model is working as expected, it is finally time to measure the model performance on the test set. Once we measure the test set performance, we report it whatever value it may be, and NOT go back and refine the model any further.

   Hint: Be sure to vectorize your code as much as possible! Training can be very slow otherwise.

   Solution : Test accuracy: 0.9486. The related code and test accuracy in the OmerKoc_PS4_Q3.ipynb file.