IITM-CS5691 : Pattern Recognition and Machine Learning
Assignment 2 (MKN Section)

Release Date: Nov 3, 2021
Due Date : Nov 12, 23:59

Roll No: ME18B181 Name: Aniruddha Gandhewar
Roll No: ME18B182 Name: Vasudev Gupta
Team name:
References (if any):

- This assignment has to be completed in teams of two. Collaborations outside the team are strictly prohibited.
- Use LaTeX to write-up your solutions (in the solution blocks of the source LaTeX file of this assignment), and submit the resulting single pdf file at GradeScope by the due date. (Note: **No late submissions** will be allowed, other than one-day late submission with 10% penalty or four-day late submission with 30% penalty! Within GradeScope, indicate the page number where your solution to each question starts, else we won't be able to grade it!)
- For the programming questions, please submit your code directly in moodle (carefully following the filename/folder/README conventions given in the questions/moodle), but provide your results/answers/Colab-link in the pdf file you upload to GradeScope. We will run plagiarism checks on codes, and any detected plagiarism in writing/code will be strictly penalized. **Please ensure that the link to your Colab notebook/code is private and give access to all TAs.**
- If you have referred a book or any other online material for obtaining a solution, please cite the source. Again don't copy the source *as is* - you may use the source to understand the solution, but write-up the solution in your own words.
- Points will be awarded based on how clear, concise and rigorous your solutions are, and how correct your code is. Overall points for this assignment would be **min(your score including bonus points scored, 60)**.
- Check the Moodle discussion/announcement forums regularly for updates regarding the assignment. Please start early and clear all doubts ASAP. Post your doubt only on Moodle Discussion Forum so that everyone is on the same page. Please note that the TAs can **only** clarify doubts regarding problem statements (they won't discuss any prospective solution or verify your solution or give hints).

1. (15 points) [SVM]

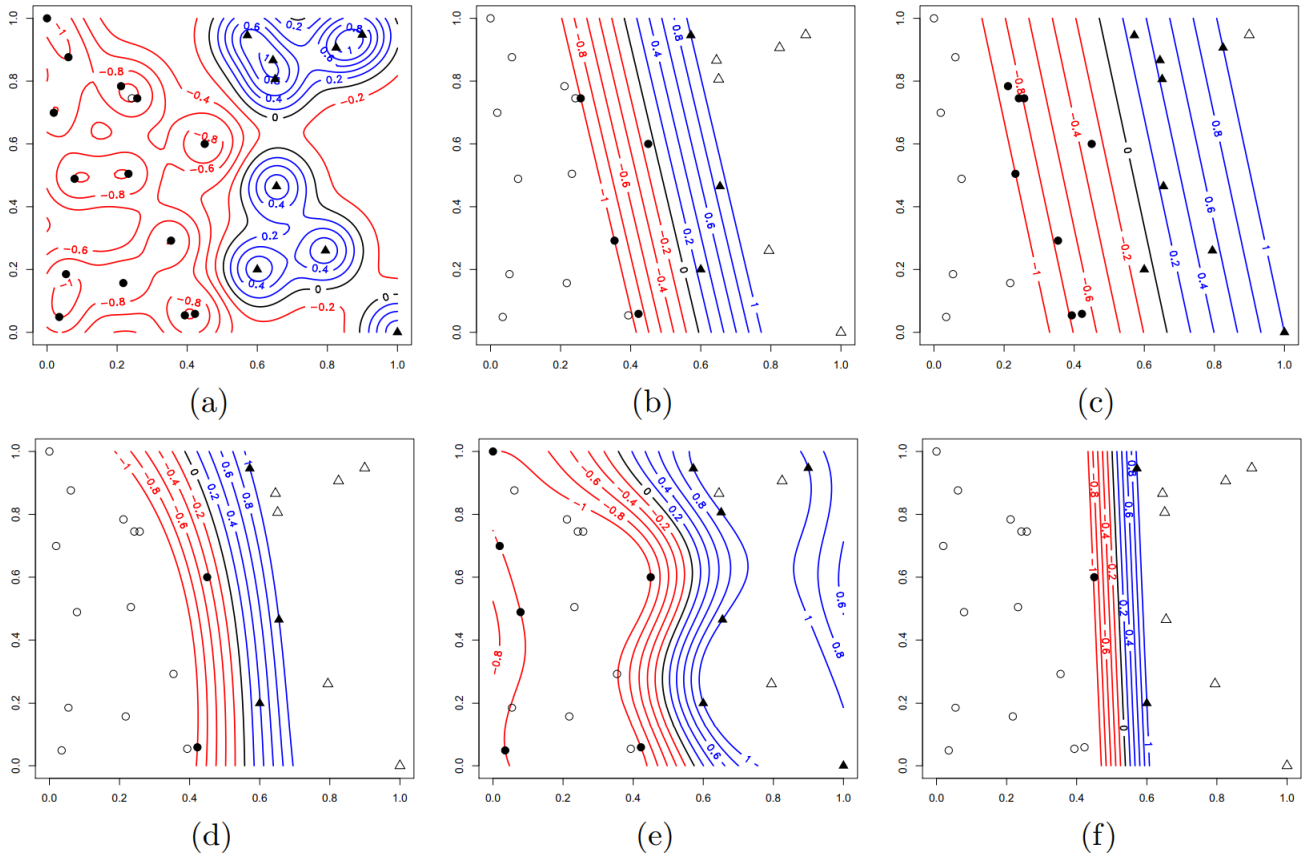
   (a) (3 points) A Gaussian or Radial Basis Function (RBF) kernel with inverse width $k > 0$ is

   $$K(u, v) = e^{-k||u-v||^2}.$$

   Below figures show decision boundaries and margins for SVMs learned on the exact same dataset. Parameters used for the different runs are as follows:
   i) Linear Kernel with C = 1
   ii) Linear Kernel with C = 10
   iii) Linear Kernel with C = 0.1

iv) RBF Kernel with k = 1, C = 3
v) RBF Kernel with k = 0.1, C = 15
vi) RBF Kernel with k = 10, C = 1

Find out which figure plot would have resulted after each run mentioned above. Justify your answer.



Circle and Triangles denotes class 1 and class 2 respectively, solid points are support vectors.

Solution:
a) RBF Kernel with k = 10, C = 1
b) Linear Kernel with C = 1
c) Linear Kernel with C = 0.1
d) RBF Kernel with k = 0.1, C = 15
e) RBF Kernel with k = 1, C = 3
f) Linear Kernel with C = 10

Reasons:

- The hyper-parameter C is used to weight the penalty contributed by points that violate the margin. An large value of C results in a model that classifies points accurately but is susceptible to small changes in the training data and thus has high variance. While, a small value of C results in a simpler model which is able to generalize well but sacrifices its accuracy.
- The hyper-parameter k is a measure of the influence of a single training sample in the RBF kernel. Low values indicate a small neighborhood while high values indicate a much larger neighborhood . The hyper-parameter k can also be thought of as the radius of influence of samples chosen as support vectors by the model. As a result, concentric circles form with very small k values.

(b) (12 points) Consider $(x_1, y_1), ..., (x_n, y_n)$ as a training data where $x_i \in \mathbb{R}^m, y_i \in \mathbb{R}$. Epsilon-sensitive loss function for regression is given below:

$$L_\epsilon(x, y, f) = |y - f(x)|_\epsilon = max(0, |y - f(x)| - \epsilon).$$

Here x is the input, y is the output, and f is the function used for predicting the label. The cost function of Support Vector Regression or SVR is:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} L_\epsilon(x_i, y_i, f)$$

where $f(x) = w^T x$, and $C, \epsilon > 0$ are parameters. Now answer these questions on SVR, with expressions simplified as much as possible.

i. (2 points) What happens when $C \to \infty$? Write the primal form of SVR for this case. (Hint: You may use two constraints per data point to capture the modulus operation.)

Solution:
When $C \to \infty$

$$\Rightarrow \quad L_\epsilon(x_i, y_i, f) = 0 \quad \forall i \in 1, 2, 3, ...N$$
$$\Rightarrow \quad |y_i - f(x_i)| < \epsilon \quad \forall i \in 1, 2, 3, ..N$$

Hence, we will get the following as the primal form of SVR:

$$\begin{aligned} &\min_w \quad \frac{1}{2}\|w\|^2 \\ &\text{such that} \\ &|y_i - w^T x_i| < \epsilon \quad \forall i \in 1, 2, 3, ..N \end{aligned} \tag{1}$$

ii. (2 points) The rest of the questions are for any general value of C. Extend the above primal form for SVR to handle any general C using appropriate slack variables.
(Hint: Try two slack variables per data point in SVR, instead of the one in SVM.)

Solution: In the above cost function, ϵ defines the region inside which errors are ignored. We can introduce slack variables ξ and ξ^* to account for errors in points that lie outside the ϵ as follows.

$$\begin{aligned} y_i - w^T x_i - \epsilon &\leqslant \xi_i \\ w^T x_i - y_i - \epsilon &\leqslant \xi_i^* \\ \xi_i, \xi_i^* &\geqslant 0; \quad \forall i \in 1, 2, 3, ..N \end{aligned} \tag{2}$$

Thus, we can write primal form as following:

$$\min_{w, \xi_i \in R^n, \xi_i^* \in R^n} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

such that

$$y_i - w^T x_i - \epsilon \leqslant \xi_i$$

$$w^T x_i - y_i - \epsilon \leqslant \xi_i^*$$

$$\xi_i, \xi_i^* \geqslant 0; \quad \forall i \in 1, 2, 3, ..N$$

iii. (2 points) Provide the Lagrangian function for the above primal. Can this function take infinite values, and if so under what conditions?

Solution: Lagrangian function can be calculated as follows:

$$L(w, \xi^*, \xi, \alpha^*, \alpha, \beta^*, \beta) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}(\xi + \xi^*) - \sum_{i=1}^{N}\alpha_i(\epsilon + \xi - y_i + w^T x_i) -$$

$$\sum_{i=1}^{N}\alpha_i^*(\epsilon + \xi^* + y_i - w^T x_i) - \sum_{i=1}^{N}\beta_i(\xi) - \sum_{i=1}^{N}\beta_i^*(\xi^*)$$

After simplifying further, we get:

$$\begin{aligned} L =& \frac{1}{2}\|w\|^2 + \sum_{i}^{N}(\alpha_i^i - \alpha_i)w^T x_i + \sum_{i=1}^{N}\xi(C - \alpha_i - \beta_i) \\ &+ \sum_{i=1}^{N}\xi^*(C - \alpha_i^* - \beta_i^*) - \sum_{i=1}^{N}\epsilon(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N}y_i(\alpha_i - \alpha_i^*) \end{aligned}$$

The above Lagrangian will take ∞ approaching values only under the following conditions:

1. $C - \alpha_i - \beta_i \neq 0$ and $\xi \to \infty$
2. $C - \alpha_i^* - \beta_i^* \neq 0$ and $\xi^* \to \infty$

iv. (2 points) Derive the Dual function from the above Lagrangian. Can this function take infinite values, and if so under what conditions?

Solution: Derivation of the Dual Function from the Lagrangian shown in previous part:

$$L_{Dual} = \min_{w,\xi^*,\xi} L$$

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) x_i = 0$$

$$\Rightarrow w = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) x_i$$

Following conditions must be satisfied, else the minimisation objective will $\to \infty$ and the dual function would not exist.

$$\begin{aligned} C - \alpha_i - \beta_i = 0 \quad \forall i \in 1, 2, 3, ..., N \\ C - \alpha_i^* - \beta_i^* = 0 \quad \forall i \in 1, 2, 3, ..., N \end{aligned} \tag{3}$$

After putting the value of w obtained in equation 1, we get the following:

$$L_{Dual} = -\frac{1}{2} \sum_{i,j=1}^{N} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) x_i^T x_j - \sum_{i=1}^{N} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{N} y_i (\alpha_i - \alpha_i^*)$$

v. (2 points) Give the dual form of SVR, and comment on whether quadratic optimization solvers can be used to solve the dual problem?

Solution: SVR dual form is formulated as following:

$$\max_{\alpha_i \geqslant 0, \alpha_i^* \geqslant 0} -\frac{1}{2} \sum_{i,j=1}^{N} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) x_i^T x_j - \sum_{i=1}^{N} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{N} y_i (\alpha_i - \alpha_i^*)$$

Following conditions are necessary for existence of dual form:

$$\begin{aligned} C - \alpha_i - \beta_i = 0 \quad \forall i \in 1, 2, 3, ..., N \\ C - \alpha_i^* - \beta_i^* = 0 \quad \forall i \in 1, 2, 3, ..., N \end{aligned} \tag{4}$$

The problem has a quadratic objective with linear constraints, hence it can be solved by a quadratic optimization solver.

vi. (2 points) Write the KKT conditions that can be used to link the primal and dual solutions.

Solution: KKT Conditions as follows:

$$\alpha_i(\epsilon + \xi - y_i + w^T x_i) = 0$$

$$\alpha_{i*}(\epsilon + \xi^* + y_i - w^T x_i) = 0$$

$$y_i - w^T x_i - \epsilon \leqslant \xi_i$$

$$w^T x_i - y_i - \epsilon \leqslant \xi_i^*$$

$$\alpha_i, \alpha_i^* \geqslant 0$$

$$\beta_i \xi_i = 0$$

$$\beta_i^* \xi_i^* = 0$$

The above equations holds for all $i \in 1, 2, 3, ..., N$

2. (15 points) [ANN]

(a) (3 points) Consider the Artificial Neural Network (ANN) in figure 1 involving a single hidden neuron for solving the XOR problem. Show that the network solves the XOR problem by constructing decision regions, and a truth table for the network.

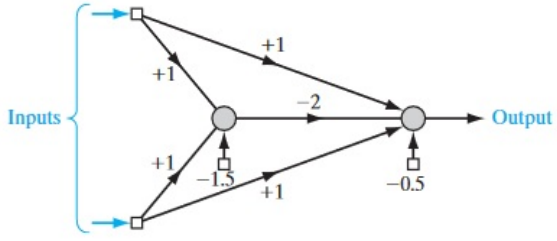


Figure 1: ANN for XOR problem

Solution:

We consider the step activation function (output is 0 when the input is negative and 1 otherwise) in all the layers.

We are considering x_1 and x_2 as the inputs to the artificial neural network (ANN), f to the activation function and a^* to the output of the hidden layer neuron.

$$a^* = f(x_1 + x_2 - 1.5)$$

Now if $x_1 + x_2 \geqslant 1.5 => a = 1$. This will be possible only when both x1 and x2 are 1.

Also,

$$y = f(x_1 + x_2 - 2a - 0.5)$$

Now if $x_1 + x_2 < 1.5$, then $a = 0$. Therefore, above equation will become this:

$$y = f(x_1 + x_2 - 0.5)$$

If $x_1 + x_2 < 0.5 => y = 0$. This will happen only when $x_1 = 0$ & $x_2 = 0$

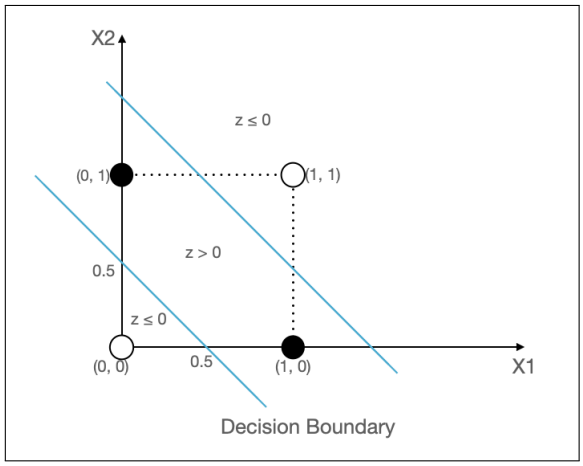
In other cases i.e $(0, 1), (1, 0)$, $y = 1$. Therefore, it solves the XOR problem.

Following figure shows the truth table for the XOR network:

x1	x2	output
0	0	0
1	1	0
1	0	1
0	1	1

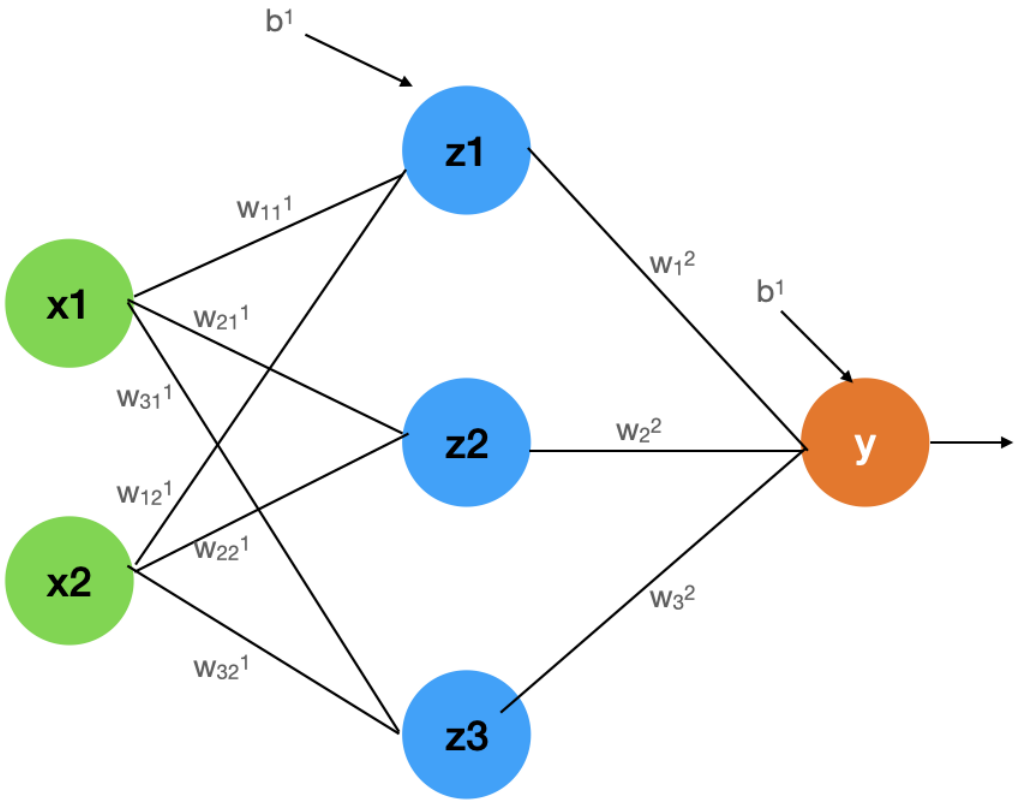
Truth Table

Following figure shows the decision boundary for XOR network:



(b) (3 points) Show, for a feed-forward neural network (FFN) with **tanh** hidden unit activation functions and a sum-of-squares error function, that the origin in weight space is a stationary point of the error function.

Solution: Consider a two layered-network with m neurons in the hidden layer (shown 3 in below figure).



Single forward pass through above model will look like this:

$$z = W^{(1)}x + b^{(1)}$$

$$a = \tanh(z)$$

$$\hat{y} = W^{(2)}a + b^{(2)}$$

$$L = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

The partial derivative of the L with respect to $w_{mn}^{(1)}$ (a general parameter in $W^{(1)}$):

$$\frac{\partial L}{\partial w_{mn}^{(1)}} = \sum_{i=1}^{N} 2(y_i - \hat{y}_i)\frac{\partial \hat{y}_i}{\partial w_{mn}^{(1)}}$$

$$\frac{\partial \hat{y}_i}{\partial w_{mn}^{(1)}} = \frac{\partial \hat{y}_i}{\partial a_m}\frac{\partial a_m}{\partial w_{mn}^{(1)}} = w_m^{(2)}\frac{\partial a_m}{\partial w_{mn}^{(1)}}$$

$$\frac{\partial a_m}{\partial w_{mn}^{(1)}} = f'(z_m)\frac{\partial z_m}{\partial w_{mn}^{(1)}} = f'(z_m)x_n$$

$$\Rightarrow \frac{\partial L}{\partial w_{mn}^{(1)}} = \sum_{i=1}^{N} 2(y_i - \hat{y_i}) w_m^{(2)} f'((\sum_{i=1}^{N} x_n w_{mi}) + b_m^{(1)}) x_n$$

Using above equations we conclude that, if $w_m^{(2)} = 0$, the gradients $\rightarrow 0$. This will happen similarly for bias (b). Now, after finding the gradients with respect to parameters of the first layer, we get:

$$\frac{\partial L}{w_m^{(2)}} = f(\sum_{i=1}^{N} w_{mi} x_i + b_m)$$

If all weights and biases are zeros, $\sum_{i=1}^{N} w_{mi} x_i + b_m$ will also become zero which will make tanh(0) = 0 and hence making the gradient with respect to parameters of the first layer equal to zero. Hence, we can say that origin in the weight space is the stationary point.

(c) (2 points) Sigmoid is one of the popular activation functions used in ANNs. Let's understand the drawbacks of using sigmoid as an activation function.

- **Sigmoid function causes gradients to vanish.** When is this statement true? Justify your answer clearly.
- **If a Sigmoid function is not zero-centered,** then what could be the possible consequence of this – can it prohibit efficient training of the ANN?

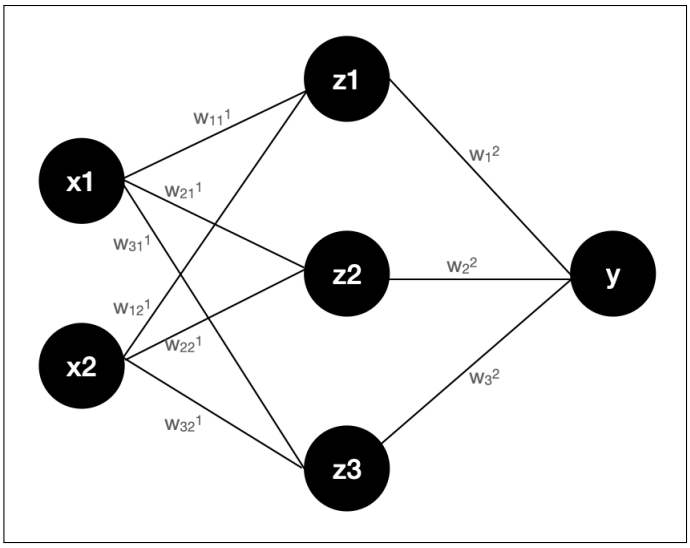
Solution:
Sigmoid function causes gradients to vanish problem: Yes, It's true that sigmoid function causes gradients to vanish. This is because $\sigma'(x) = \sigma(x) \times (1 - \sigma(x))$ and as x becomes larger, $\sigma(x) \rightarrow 1$, hence $\sigma'(x) \rightarrow 0$ Similarly, as x becomes smaller, $\sigma(x) \rightarrow 0$, hence $\sigma'(x) \rightarrow 0$. This problem worsens as we increase the number of layers in a deep neural network since gradients of initial layers depends on gradients of later layers (because of back-propogation). Hence, this is a vanishing gradients problem.

If a Sigmoid function is not zero-centered problem: Since the outputs of the sigmoid are always between 0 and 1, hence gradients will be either positive or negative during gradient descent. As a consequence, the gradient updates are too far in a particular direction and hence optimization becomes harder. Hence, we may need more number of iterations to get the good model.

(d) (2 points) Considering a simple ANN with two input neurons, 3 neurons in a single hidden layer and 1 output neuron. Assume that sigmoid is used as an activation function.

- Mathematically show why initializing all weights with the same value is a bad idea?
- What could be better strategies for weight initialisation for the above ANN architecture and why?

Solution: a)



$$z_1 = \sigma(w_{11}^1 x1 + w_{12}^1 + b^1)$$
$$z_2 = \sigma(w_{21}^1 x1 + w_{22}^1 + b^1)$$
$$z_3 = \sigma(w_{31}^1 x1 + w_{32}^1 + b^1)$$

$$y = \sigma(w_1^2 z_1 + w_2^2 z_2 + w_3^2 z_3 + b^2)$$

Now, when

$$w_{11}^1 = w_{12}^1 = w_{21}^1 = w_{22}^1 = w_{31}^1 = w_{32}^1 = w_1^2 = w_2^2 = w$$

$$b^1 = b^2 = b$$

We get,

$$z_1 = z_2 = z_3 = z$$
$$y = \sigma(3wz + b)$$
$$y = \sigma(Wz + b)$$

For the above equation, we can clearly say that regardless of increasing number of neurons in the hidden layer, if all the weights are initialized to same value, all the hidden neurons

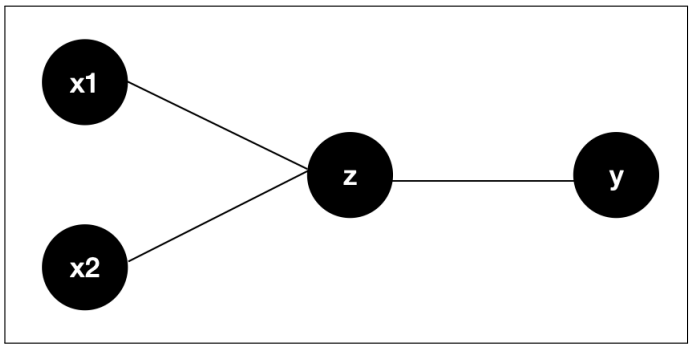
will work just like a single neuron. Hence, model won't as complex as we would want and this may lead to underfitting. Also, most the weights remain same during training as we derived in following part:

$$\frac{\partial y}{\partial w_1^2} = y(1-y)z_1$$
$$\frac{\partial y}{\partial w_2^2} = y(1-y)z_2$$
$$\frac{\partial y}{\partial w_3^2} = y(1-y)z_3$$

Since $z_1 = z_2 = z_3 = z$,

$$\frac{\partial y}{\partial w_1^2} = \frac{\partial y}{\partial w_2^2} = \frac{\partial y}{\partial w_3^2}$$

Similarly for $w_{11}^1, w_{21}^1, w_{31}^1$ (cluster-1) and $w_{12}^1, w_{22}^1, w_{32}^1$ (cluster-2).
Hence all the weights in the 2nd layer continue to remain same and weights in first layer will be clustered into 2 group and each group will have same weights. Above complex architecture can be easily simplified into the following architecture:



b) Random initialization or Xavier initialization scheme would be the better strategy for initializing weights. Biases are usually set to 0s for simplicity.

(e) (5 points) In the planet Pandora, there is a creature called "Direhorse". The following table shows the **weights of eye lenses of direhorses as a function of age**. No simple analytical function can exactly interpolate these data because we do not have a single-valued function. Instead, we have a nonlinear least squares model of this data set, using a negative exponential , as described by

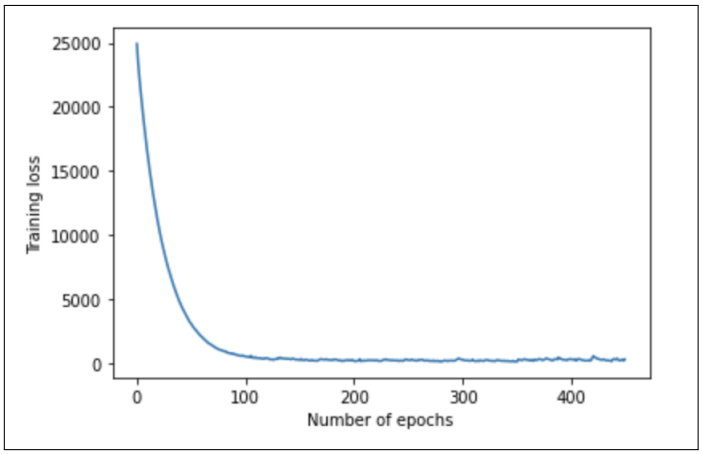
$$y = 233.846 * (1 - exp(-0.006042x)) + \epsilon$$

where ϵ is an error term.
You are appointed as the chief data scientist on the planet and the department of astrobiology has requested your assistance. Your task is briefly described below :
Using the back-propagation algorithm, design a feed-forward neural network (FFN, aka multilayer perceptron) that provides a nonlinear least-squares approximation to this dataset. Compare your results against the least-squares model given above. Briefly comment on when your designed FFN led to underfitting or overfitting, and how you fixed these issues.

Ages (days)	Weights (mg)	Ages (days)	Weights (mg)	Ages (days)	Weights (mg)	Ages (days)	Weights (mg)
15	21.66	75	94.6	218	174.18	338	203.23
15	22.75	82	92.5	218	173.03	347	188.38
15	22.3	85	105	219	173.54	354	189.7
18	31.25	91	101.7	224	178.86	357	195.31
28	44.79	91	102.9	225	177.68	375	202.63
29	40.55	97	110	227	173.73	394	224.82
37	50.25	98	104.3	232	159.98	513	203.3
37	46.88	125	134.9	232	161.29	535	209.7
44	52.03	142	130.68	237	187.07	554	233.9
50	63.47	142	140.58	246	176.13	591	234.7
50	61.13	147	155.3	258	183.4	648	244.3
60	81	147	152.2	276	186.26	660	231
61	73.09	150	144.5	285	189.66	705	242.4
64	79.09	159	142.15	300	186.09	723	230.77
65	79.51	165	139.81	301	186.7	756	242.57
65	65.31	183	153.22	305	186.8	768	232.12
72	71.9	192	145.72	312	195.1	860	246.7
75	86.1	195	161.1	317	216.41		

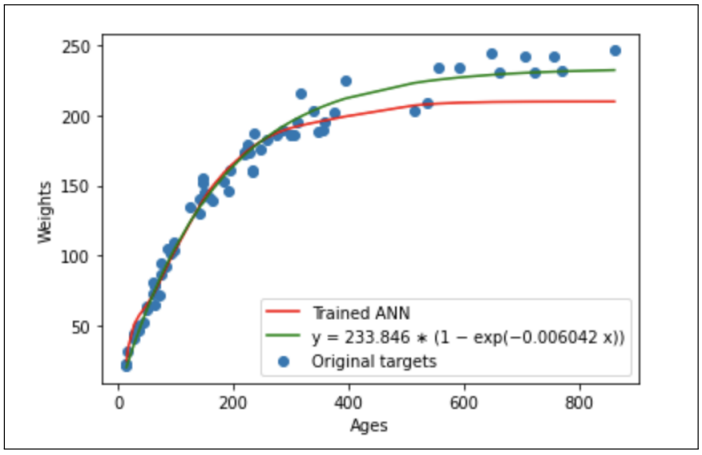
Paste the training loss plot and report your results in the submissions.
Note: You can use any python library. **Share the link to your Colab notebook with appropriate output already displayed and make sure the notebook is private, with access given to TAs.**

Solution: **Link to colab notebook**
We have plotted the curve showing training loss collected while training the model.



From above plot, we can clearly see that training loss is constantly decreasing and model is learning something useful from this data.

Futher, we have plotted the distribution of data, predictions using least-square model given in question, and the deep learning model.



We found that our neural network was able to approximate most the data and is performing very similar to the least-square model. Our model is slightly overfitting the smaller ages as more data is having small ages.

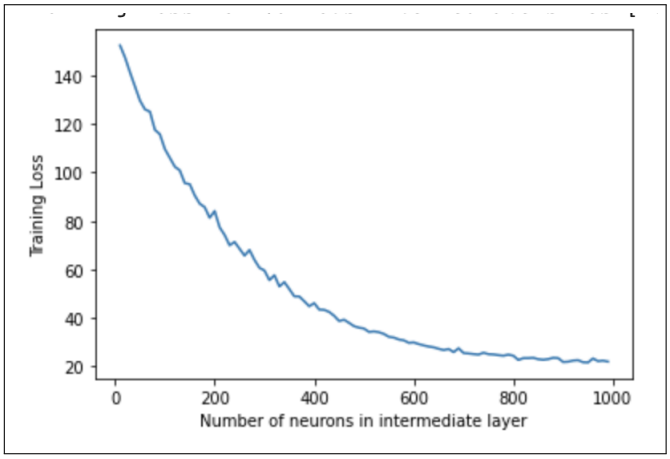
Having very less number of neurons in the intermediate layer, and very shallow network (very few layers) and larger learning rate leads to divergence and hence was resulting in underfitting. To handle these issues, we made network enough deeper and experimented by increasing number of neurons in the intermediate layers.

(f) (5 points) [OPTIONAL BONUS] ANN python libraries are banned in Pandora during the time of this project, and you are forced instead to use only the numpy library and any plotting

libraries for this task. Code from scratch the backpropagation algorithm for a FFN with a single hidden layer but with variable number of hidden neurons, and report how the network performance is affected by varying the size of the hidden layer for the Direhorse dataset. Share the link to your Colab notebook with appropriate output already displayed and make sure the notebook is private, with access granted to TAs.

Solution: **Link to colab notebook**

In the following plot, we have tried to analyze the impact of increasing the number of neurons in the intermediate layers on the training loss.



3. (15 points) [BOOST IS THE SECRET OF...]

   (a) (6 points) [ADABOOST PEN/PAPER] Say we have the following toy dataset in the form: $x_1 : (0, -1)$ $\texttt{label} : (-)$,$x_2 : (1, 0)$ $\texttt{label} : (+)$,$x_1 : (-1, 0)$ $\texttt{label} : (+)$,$x_1 : (0, 1)$ $\texttt{label} : (-)$.

      i. (2 points) Using decision stumps as weak classifiers show how Adaboost works. Compute the parameters at each timestep. Compute upto $T = 4$.

Solution:

> **Note**: As the number of training samples is small, we will sample with replacement upto 4 times the total number of samples. We do this to ensure the fraction of each sample is as close to its probability represented by its weight.

Iteration 1

Initialized Weights

w_1	w_2	w_3	w_4
0.25	0.25	0.25	0.25

Using the sampled data, we train the Decision Tree Classifier ($h_1(x)$).

$$h_1(x) = \begin{cases} +1 & \text{if } x_1 < -0.5 \\ -1 & \text{if } x_1 \geqslant -0.5 \end{cases}$$

Performance of $h_1(x)$ on our training sample is summarized in the tables below.

Table 1

Feature 1	**Feature 2**	**Target**	**Prediction**
0	1	-1	-1
0	1	-1	-1
1	0	1	-1
0	-1	-1	-1
-1	0	1	1
-1	0	1	1
0	1	-1	-1
0	-1	-1	-1
-1	0	1	1
-1	0	1	1
0	-1	-1	-1
0	-1	-1	-1
-1	0	1	1
0	-1	-1	-1
0	-1	-1	-1
0	-1	-1	-1

Error	0.25
α_1	0.5493

Iteration 2

Updated Weights

w_1	w_2	w_3	w_4
0.1667	0.50	0.1667	0.1667

Using the re-sampled data, we train the Decision Tree Classifier ($h_2(x)$).

$$h_2(x) = \begin{cases} +1 & \text{if } x_2 < 0.5 \\ -1 & \text{if } x_2 \geqslant 0.5 \end{cases}$$

Performance of $h_2(x)$ on our training sample is summarized in the tables below.

Table 2

Feature 1	**Feature 2**	**Target**	**Prediction**
0	-1	-1	1
1	0	1	1
1	0	1	1
1	0	1	1
1	0	1	1
1	0	1	1
1	0	1	1
-1	0	1	1
0	1	-1	-1
1	0	1	1
0	1	-1	-1
1	0	1	1
-1	0	1	1
-1	0	1	1
1	0	1	1
1	0	1	1

Error	0.1667
α_2	0.8047

Iteration 3

Updated Weights

w_1	w_2	w_3	w_4
0.50	0.30	0.10	0.10

Using the re-sampled data, we train the Decision Tree Classifier ($h_3(x)$).

$$h_3(x) = \begin{cases} +1 & \text{if } x_2 > -0.5 \\ -1 & \text{if } x_2 \leqslant -0.5 \end{cases}$$

Performance of $h_3(x)$ on our training sample is summarized in the tables below.

Table 3

Feature 1	**Feature 2**	**Target**	**Prediction**
0	-1	-1	-1
1	0	1	1
0	-1	-1	-1
0	-1	-1	-1
0	-1	-1	-1
0	-1	-1	-1
0	-1	-1	-1
0	-1	-1	-1
0	1	-1	1
0	-1	-1	-1
0	1	-1	1
-1	0	1	1
-1	0	1	1
0	-1	-1	-1
0	-1	-1	-1
1	0	1	1

Error	0.2
α_3	0.6931

Iteration 4

Updated Weights

w_1	w_2	w_3	w_4
0.3846	0.2308	0.0769	0.3077

Using the re-sampled data, we train the Decision Tree Classifier ($h_4(x)$).

$$h_4(x) = \begin{cases} +1 & \text{if } x_2 > -0.5 \\ -1 & \text{if } x_2 \leqslant -0.5 \end{cases}$$

Performance of $h_4(x)$ on our training sample is summarized in the tables below.

Table 4

Feature 1	**Feature 2**	**Target**	**Prediction**
0	-1	-1	-1
1	0	1	1
0	1	-1	1
0	-1	-1	-1
0	-1	-1	-1
1	0	1	1
1	0	1	1
1	0	1	1
1	0	1	1
0	-1	-1	-1
0	-1	-1	-1
1	0	1	1
-1	0	1	1
-1	0	1	1
0	-1	-1	-1
0	-1	-1	-1

Error	0.3077
α_4	0.4054

ii. (2 points) Draw the classifier at each timestep.

Solution:

T 1

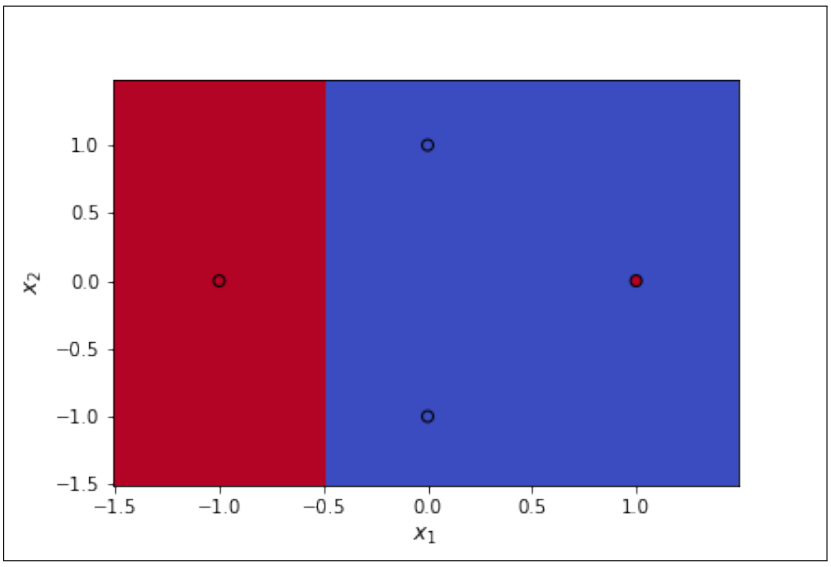


Figure 2: Decision Surface of AdaBoost Classifier (k=1)

T 2

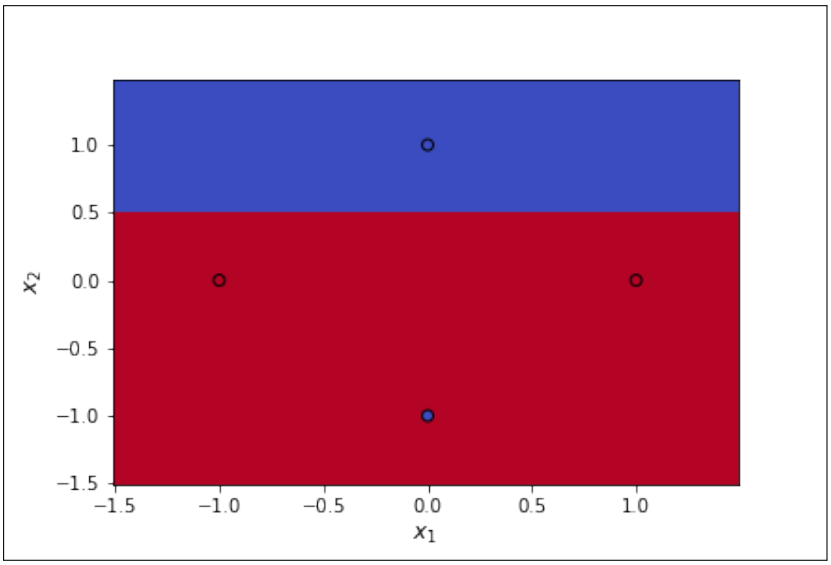


Figure 3: Decision Surface of AdaBoost Classifier (k=2)

T 3

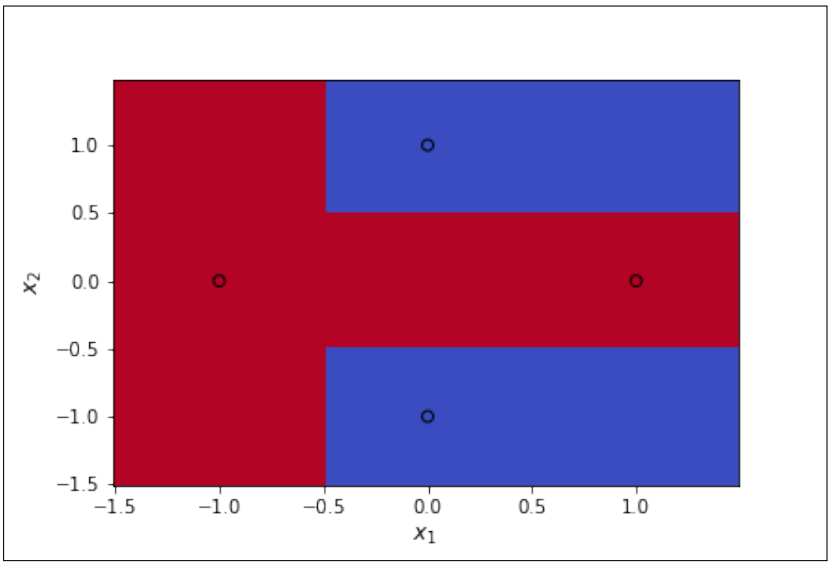


Figure 4: Decision Surface of AdaBoost Classifier (k=3)

T 4

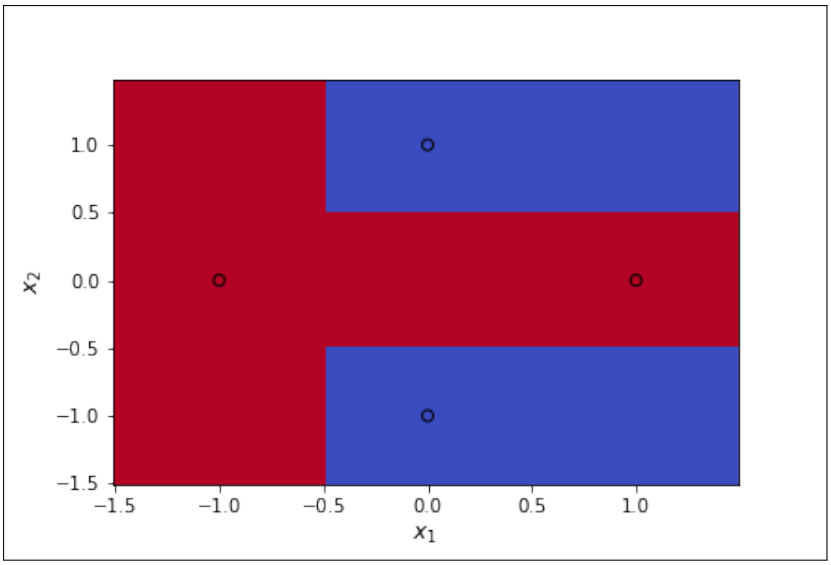


Figure 5: Decision Surface of AdaBoost Classifier (k=4)

iii. (2 points) What is the training error? Explain in short why Adaptive Boost outperforms a decision stump in this dataset.

Solution:
The following reasons are why Adaptive Boost outperforms a decision stump in this dataset.

1. The dataset is not separable using a linear decision boundary (a Decision Stump is only capable of creating a linear boundary) i.e. we need a non-linear decision boundary.

2. AdaBoost Classifier uses a weighted combination of Decision Stumps to make predictions (is able to create a non-linear decision boundary) and thus successfully classifies the training data.

Overall, AdaBoost Classifier combines multiple **"Weak Classifiers"** (Decision Stumps) into a **"Strong Classifier"**.

(b) (9 points) [ADABOOST CODE] In this question, you will code the AdaBoost algorithm from scratch. Follow the instructions in this Jupyter Notebook for this question. Find the dataset for the question here. (**NOTE: Test data should not be used for training.**)

i. (2 points) Plot the training data.

Solution:

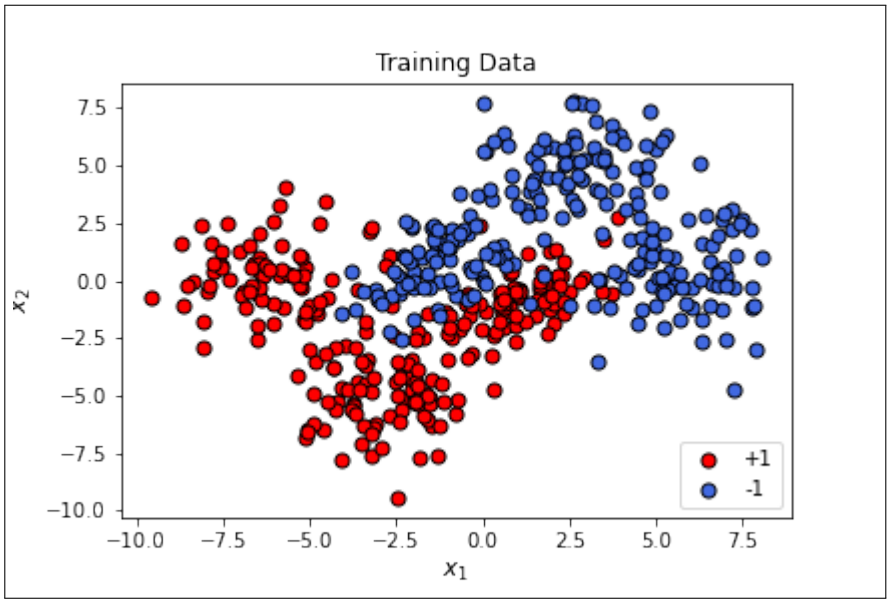


Figure 6: Plot of the training data sample

ii. (2 points) For training with k=5: (1) Plot the learnt decision surface. (2) Write down the test accuracy.

Solution:

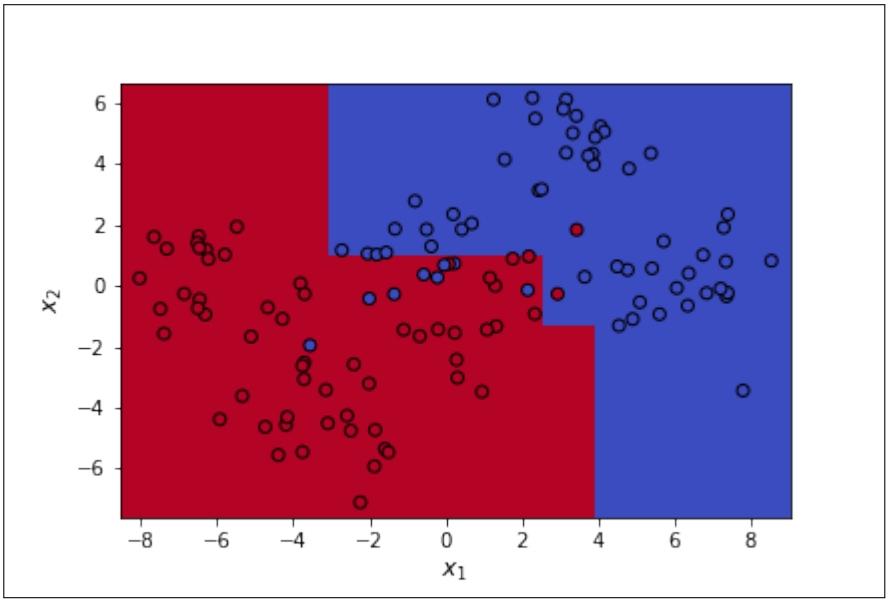


Figure 7: Decision Surface of AdaBoost Classifier (k=5)

Accuracy Score of the AdaBoost Classifier (k=5) on the Test data: **0.9076**

iii. (2 points) For training with k=100: (1) Plot the learnt decision surface. (2) Write down the test accuracy.

Solution:

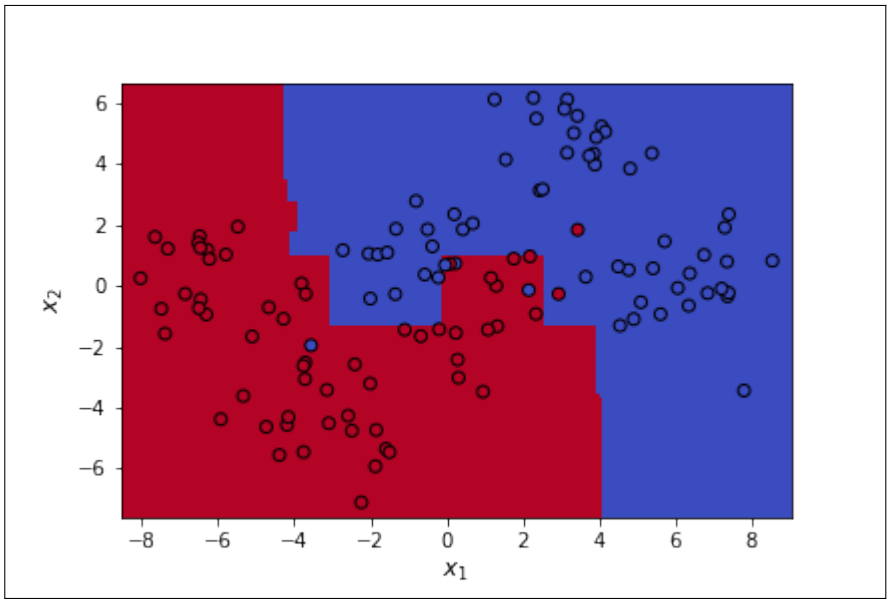


Figure 8: Decision Surface of AdaBoost Classifier (k=100)

Accuracy Score of the AdaBoost Classifier (k=100) on the Test data: **0.9496**

iv. (3 points) Paste the link to your Colab Notebook of your code. Make sure that your notebook is private and give access to all the TAs. Your notebook must contain all the codes that you used to generate the above results.

Solution: Link to colab notebook

(c) (5 points) [OPTIONAL BONUS] In the AdaBoost algorithm, you have n points, (x_i, r_i) where r_i is the class label of x_i , $i = 1, \cdots, n$, let $h_t(x)$ be the weak classifier obtained at step t, and let α_t be its weight. Recall that the final classifier is

$$H(x) = \text{sign}(f(x)) \quad f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

Show that the training error of the final classifier can be bounded from above by an exponential loss function :

$$\frac{1}{m} \sum_{i=1}^{m} I(H(x_i) \neq r_i) \leqslant \frac{1}{m} \sum_{i=1}^{i=m} e^{-f(x_i) r_i}$$

where I is the indicator function.

4. (15 points) [SINGULARLY PCA!] Consider a dataset of N points with each datapoint being a D-dimensional vector in $\mathbb{R}^D$. Let's assume that:

- we are in a high-dimensional setting where $D >> N$ (e.g., D in millions, N in hundreds).
- the $N \times D$ matrix X corresponding to this dataset is already mean-centered (so that each column's mean is zero, and the covariance matrix seen in class becomes $S = \frac{1}{N}X^TX$).
- the rows (datapoints) of X are linearly independent.

Under the above assumptions, please attempt the following questions.

(a) (6 points) Whereas X is rectangular in general, XX^T and X^TX are square. Show that these two square matrices have the same set of non-zero eigenvalues. Further, argue briefy why these equal eigenvalues are all positive and N in number, and derive the multiplicity of the zero eigenvalue for both these matrices.
(Note: The square root of these equal positive eigenvalues $\{\lambda_i := \sigma_i^2\}_{i=1,...,N}$ are called the singular values $\{\sigma_i\}_{i=1,...,N}$ of X.)

Solution:
Let A, B be two matrices, such that,

$$A : m \times n \text{ and } B : n \times m$$

then,

$$AB : m \times m,\ BA : n \times n$$

Consider,

$$C = \begin{bmatrix} tI_m & A \\ B & I_n \end{bmatrix},\ D = \begin{bmatrix} I_m & 0 \\ -B & tI_n \end{bmatrix}$$

Since C and D are square matrices, we know,

$$det(CD) = det(DC)$$
$$det \begin{bmatrix} tI_m - AB & tA \\ 0 & tI_n \end{bmatrix} = det \begin{bmatrix} tI_m & A \\ 0 & tI_n - BA \end{bmatrix}$$
$$t^n det(tI_m - AB) = t^m det(tI_n - BA)$$

Let $n > m$, then,

$$t^{n-m} det(tI_m - AB) = det(tI_n - BA)$$

where, $det(tI_m - AB)$, $det(tI_n - BA)$ represent the characteristic polynomials of AB, BA respectively.

$$\therefore \{Eigenvalues\ of\ BA\} = \{Eigenvalues\ of\ AB\} \cup (n - m)zeros$$

Now,

$$X : N \times P,\ X^T : P \times N$$

Using the above derived result, we can state that,

$$\boxed{\therefore \{\text{Eigenvalues of } X^T X\} = \{\text{Eigenvalues of } XX^T\} \cup (P - N)zeros}$$

Further, let's prove that the set of equal eigenvalues are all positive.
Consider,

$$\begin{aligned} & w^T(X^T X)w \\ &= (w^T X^T)(Xw) \\ &= (Xw)^T(Xw) \\ &= \|Xw\|^2 \end{aligned}$$

$$\boxed{\therefore\ w^T(X^T X)w\ \geqslant\ 0\ \forall\, w}$$

$\Rightarrow$ $X^T X$ is a Positive Semi-definite Matrix. We can argue the same for XX^T. All eigenvalues of a positive Semi-definite matrix are non-negative.

$\therefore$ **Set of all non-zero equal eigenvalues are Positive.**

Next, let's prove that the positive eigenvalues are N in number.
We know, XX^T is:

1. Symmetric Matrix
2. Positive Semi-definite Matrix

According to the Spectral theorem, XX^T is diagonalizable.
$\Rightarrow$ $dim\ range\ XX^T$ = No. of Eigenvectors with non-zero Eigenvalues.
Now,

$$rank\ X = dim\ range\ X = N$$

We also know,

$$range\ X \perp null\ X^T$$

and,

$$dim\ range\ X + dim\ null\ X^T = N$$

$$\therefore dim\ null\ X^T = 0$$

As a result,

$$dim\ null\ XX^T = 0 \quad (\because range\ X^T \perp null\ X)$$

Using the Fundamental Theorem of Linear Algebra, we have,

$$dim\ null\ XX^T + dim\ range\ XX^T = N$$

$$\therefore \dim \text{range } XX^T = N$$

Finally,

$$\text{No. of Eigenvectors with non} - \text{zero Eigenvalues} = N$$
$$\therefore \text{No. of non} - \text{zero Eigenvalues} = N$$

Using the above derived result,

$$\{\text{Eigenvalues of } X^TX\} = \{\text{Eigenvalues of } XX^T\} \cup (P - N)\text{zeros}$$

We can state,

$$\text{Algebraic Multiplicity of 0 } (XX^T) = 0$$
$$\text{Algebraic Multiplicity of 0 } (X^TX) = P - N$$

(b) (5 points) We can choose the set of eigenvectors $\{u_i\}_{i--=1,\ldots,N}$ of XX^T to be an orthonormal set and similarly we can choose an orthonormal set of eigenvectors $\{v_j\}_{j=1,\ldots,D}$ for X^TX. Briefly argue why this orthonormal choice of eigenvectors is possible. Can you choose $\{v_i\}$ such that each v_i can be computed easily from u_i and X alone (i.e., without having to do an eigenvalue decomposition of the large matrix X^TX; assume $i = 1, \ldots, N$ so that $\lambda_i > 0$ and $\sigma_i > 0$)?
(Note: $\{u_i\}, \{v_i\}$ are respectively called the left,right singular vectors of X, and computing them along with the corresponding singular values is called the Singular Value Decomposition or SVD of X.)

Solution:
Consider a real symmetric matrix S.
Let u_1, u_2 be eigenvectors of S and α_1, α_2 their corresponding eigenvalues, then,

$$\begin{aligned}(\alpha_1 - \alpha_2) < u_1, u_2 > &= < \alpha_1 u_1, u_2 > - < u_1, \alpha_2 u_2 > \\ &= < Su_1, u_2 > - < u_1, Su_2 > \\ &= < Su_1, u_2 > - < S^T u_1, u_2 > \\ &= < (S - S^T) u_1, u_2 > \\ &= 0\end{aligned}$$

$\therefore$ Eigenvectors with distinct Eigenvalues are orthogonal.

$\therefore$ For XX^T, we can state the following:-

1. Eigenvectors with distinct Eigenvalues are orthogonal.

2. Eigenvectors are N in number.

$\therefore$ $\{u_1, u_2, u_3, ..., u_N\}$ with $\|u_i\| = 1 \; \forall \; i$ is an orthonormal set of eigenvectors.

Yes, we can choose a v_i such that each v_i can be computed easily from u_i and X alone. Demonstrated as follows:
Consider,

$$XX^T u_i = \lambda_i u_i$$

Premultiply with X^T,

$$X^T(XX^T)u_i = \lambda_i X^T u_i$$

$$(X^T X)X^T u_i = \lambda_i X^T u_i$$

Let $X^T u_i = v_i$, then,

$$X^T X v_i = \lambda_i v_i$$

$$\therefore \; v_i \; is \; an \; eigenvector \; of \; X^T X, \; where \; X^T u_i = v_i$$

Hence, we can compute the entire set $\{v_1, v_2, v_3, ..., v_N\}$ where $X^T X v_i = \lambda_i v_i, \; \lambda_i \neq 0$.

(c) (4 points) Applying PCA on the matrix X would be computationally difficult as it would involve finding the eigenvectors of $S = \frac{1}{N} X^T X$, which would take $O(D^3)$ time. Using answer to the last question above, can you reduce this time complexity to $O(N^3)$? Please provide the exact steps involved, including the exact formula for computing the normalized (unit-length) eigenvectors of S.

Solution:
Yes, we can reduce the time complexity of PCA on X to $O(DN^2)$.

Step 1: Compute the Matrix - Matrix Adjoint Product scaled by N.

$$S' = \frac{1}{N} X X^T,$$

$$Time \; Complexity : O(DN^2)$$

Step 2: Eigenvalue Decomposition of S'.

$$S' = UDU^{-1}$$

$$Time \; Complexity : O(N^3)$$

Step 3: Compute Eigenvectors of Variance-covariance matrix $X^T X$.

$$V = X^T U$$
$$V = [v_1, v_2, v_3 ..., v_N]$$
$$\text{Time Complexity} : O(DN^2)$$

Step 4: Compute Principal Components

$$\begin{aligned} P &= XV \\ &= U\Sigma V^T V \\ &(SVD \text{ of } X) \\ &= U\Sigma \end{aligned}$$
$$\text{Time Complexity} : O(N^3)$$

Note: Step 3 can be skipped.

Overall Time Complexity: $O(DN^2)$

(d) (5 points) [OPTIONAL BONUS] Exercise 12.2 from Bishop's book helps prove why minimum value of the PCA squared error J, subject to the orthonormality constraints of the set of principal axes/directions $\{u_i\}$ that we seek, is obtained when the $\{u_i\}$ are eigenvectors of the data covariance matrix S. That exercise introduces a modified squared error $\widetilde{J}$, which involves a matrix H of Langrange multipliers, one for each constraint, as follows:

$$\widetilde{J} = Tr\left\{\widehat{U}^T S \widehat{U}\right\} + Tr\left\{H(I - \widehat{U}^T \widehat{U})\right\}$$

where $\widehat{U}$ is a matrix of dimension $D \times (D - M)$ whose columns are given by u_i. Now, any solution to minimizing $\widetilde{J}$ should satisfy $S\widehat{U} = \widehat{U}H$, and one specific solution is that the columns of $\widehat{U}$ are the eigenvectors of S, in which case H is a diagonal matrix. Show that any general solution to $S\widehat{U} = \widehat{U}H$ also gives the same value for $\widetilde{J}$ as the above specific solution.
(Hint: Show that H can be assumed to be a symmetric matrix, and then use the eigenvector expansion i.e., diagonalization of H.)