IITM-CS5691 : Pattern Recognition and Machine Learning | Release Date: Nov 3, 2021
Assignment 2 (MKN Section) | Due Date : Nov 12, 23:59

Roll No: ME18B030 | Name: S. Jeeva
Roll No: ME18B046 | Name: G. Deepak
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]

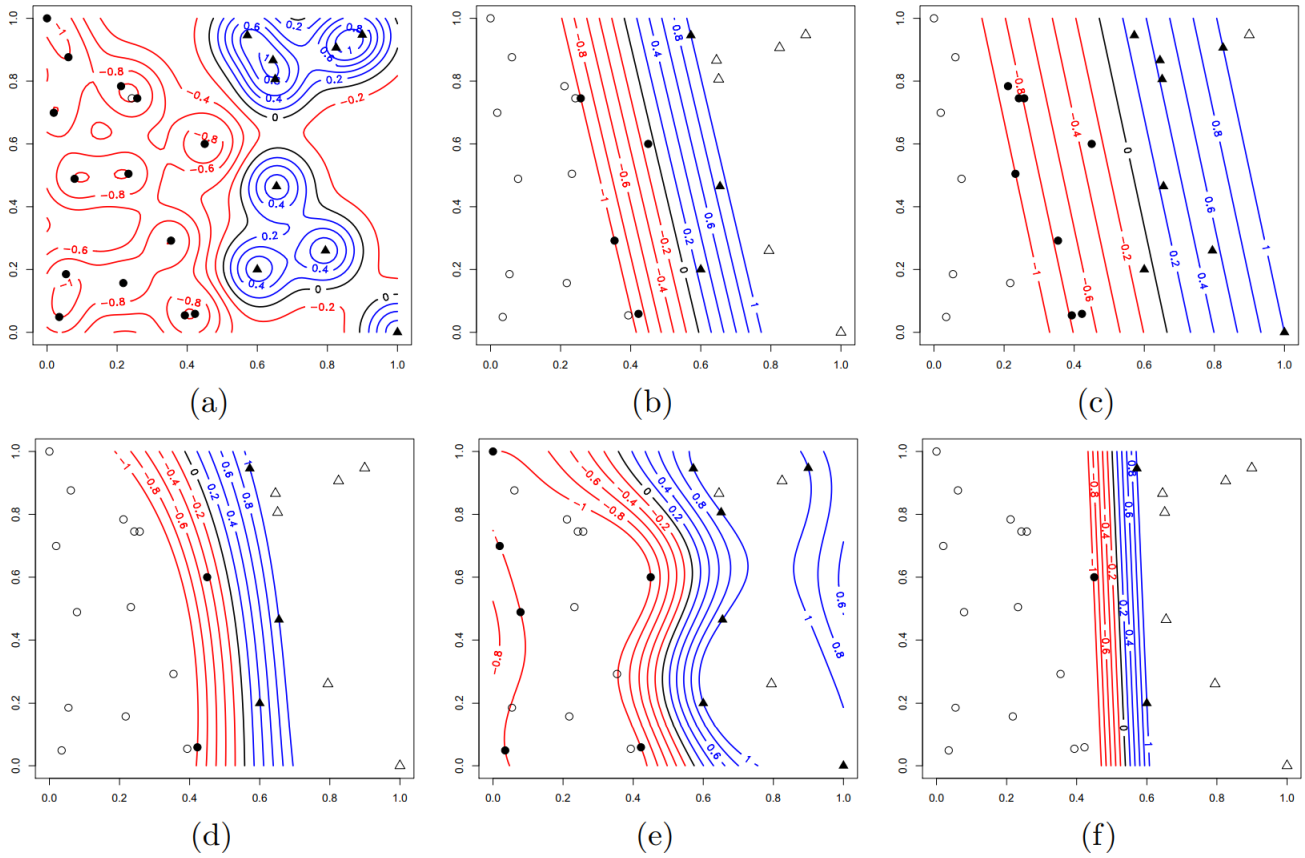
   (a) (3 points) A Gaussian or Radial Basis Function (RBF) kernel with inverse width $k > 0$ is

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

   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:
Linear Kernel assumes the data is linearly separable and uses a linear hyper plane to classify the data, as in the figures b,c and f.
On the other hand, RBF kernel uses the kernel $K(X_1, X_2) = \exp\left(-\frac{\|X_1 - X_2\|^2}{2\sigma^2}\right)$ which leads to non-linear hyper planes, as in the figures a,d and e.
Regularisation for the classifiers is decided based on the value of C, larger values of C chooses smaller margin, which results in the following:

i) Linear Kernel with C = 1 $\rightarrow$ plot(b)

ii) Linear Kernel with C = 10 $\rightarrow$ plot(f)
iii) Linear Kernel with C = 0.1 $\rightarrow$ plot(c)
iv) RBF Kernel with $k = 1$, C = 3 $\rightarrow$ plot(e)
v) RBF Kernel with $k = 0.1$, C = 15 $\rightarrow$ plot(d)
vi) RBF Kernel with $k = 10$, C = 1 $\rightarrow$ plot(a)

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

Solution:
We have an ϵ-sensitive error function, having a linear cost, given by,

$$L_\epsilon(x, y, f) = \begin{cases} 0, & |y - f(x)| \leqslant \epsilon; \\ |y - f(x)| - \epsilon, & |y - f(x)| > \epsilon \end{cases}$$

$$L_\epsilon(x, y, f = w^T x) = \begin{cases} 0, & |y - w^T x| \leqslant \epsilon; \\ |y - w^T x| - \epsilon, & |y - w^T x| > \epsilon \end{cases}$$

When $C \rightarrow \infty$ we have the loss function to be zero and this gives rise to 2 inequalities by the virtue of $|y - w^T x| \leqslant \epsilon$. In this case we don't need slack variables because

$C \to \infty$. This gives rise to the two inequalities from the modulus operation as,

$$y_i - w^T x_i - \epsilon \leqslant 0$$

$$-y_i + w^T x_i - \epsilon \leqslant 0$$

From the above inequalities and given the fact that $C \to \infty$ we can write the primal form of SVR for this case as,

$$\min_w \frac{1}{2}||w||^2$$

such that

$$y_i - w^T x_i - \epsilon \leqslant 0$$
$$-y_i + w^T x_i - \epsilon \leqslant 0$$

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:
Introducing Slack variables ξ_1 and ξ_2. We need two slack variables to correspond to each inequality arising from the modulus operation in the loss function $L_\epsilon(x, y, f)$. Since C takes any general value we can consider the second case where the loss function is $max(|y - f(x)| - \epsilon)$ by the virtue of $|y - f(x)| \geqslant \epsilon$, because this value of the parameter C strictly allows only zero loss L_ϵ.

We can then write the inequalities with the slack variables (where ξ_{1i} and ξ_{2i} are both $\geqslant 0 \; \forall i = 1 \ldots n$) as,

$$\xi_{1i} \geqslant y_i - w^T x_i - \epsilon$$
$$\xi_{2i} \geqslant -y_i + w^T x_i - \epsilon$$

The set of inequalities can be rearranged and written as,

$$y_i - w^T x_i - \epsilon - \xi_{1i} \leqslant 0$$

$$-y_i + w^T x_i - \epsilon - \xi_{2i} \leqslant 0$$

Hence, the primal form for SVR for a general C can be written as,

$$\min_{w, \xi_1, \xi_2} \frac{1}{2}||w||^2 + C \sum_{i=1}^{n} (\xi_{1i} + \xi_{2i})$$

such that

$$
\begin{aligned}
& y_i - w^T x_i - \epsilon - \xi_{1i} \leqslant 0 \\
& -y_i + w^T x_i - \epsilon - \xi_{2i} \leqslant 0 \\
& \xi_{1i} \geqslant 0 \implies -\xi_{1i} \leqslant 0 \\
& \xi_{2i} \geqslant 0 \implies -\xi_{2i} \leqslant 0
\end{aligned}
$$

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

Solution:
To write the lagrangian function we need four Karush-Kuhn-Tucker multipliers α_1, α_2, β_1, β_2 for the corresponding four inequalities in the primal form. Thus, we can write the lagrangian form as,

$$
\begin{aligned}
L(w, \xi_1, \xi_2, \alpha_1, \alpha_2, \beta_1, \beta_2) = \min_{w, \xi_1, \xi_2} \quad & \frac{1}{2}||w||^2 + C \sum_{i=1}^{n} (\xi_{1i} + \xi_{2i}) + \sum_{i=1}^{n} (-\beta_{1i}\xi_{1i}) \\
& + \sum_{i=1}^{n} (-\beta_{2i}\xi_{2i}) + \sum_{i=1}^{n} \alpha_{1i} \left(y_i - w^T x_i - \epsilon - \xi_{1i}\right) \\
& + \sum_{i=1}^{n} \alpha_{2i} \left(-y_i + w^T x_i - \epsilon - \xi_{2i}\right)
\end{aligned}
$$

For feasibility of the above expressed lagrangian function we need values of the KKT multipliers and other variables that ensure that the lagrangian function can take infinite values. This is possible under the following conditions,

$$
\begin{aligned}
& C - \beta_{1i} - \alpha_{1i} = 0 \ \forall\, i = 1 \ldots n \\
& C - \beta_{2i} - \alpha_{2i} = 0 \ \forall\, i = 1 \ldots n \\
& w - \sum_{i=1}^{n} (\alpha_{1i} - \alpha_{2i})\, x_i = 0 \\
\implies & w = \sum_{i=1}^{n} (\alpha_{1i} - \alpha_{2i})\, x_i = 0 \\
& w = x^T(\alpha_1 - \alpha_2))
\end{aligned}
$$

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

Solution: From the above feasibility conditions of the lagrangian, we can write the dual function as,

$$\Phi(\alpha_1, \alpha_2, \beta_1, \beta_2) = \min_{w} \quad \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \epsilon\,(\alpha_{1i} + \alpha_{2i}) + \sum_{i=1}^{n} y_i\,(\alpha_{1i} - \alpha_{2i}) - \sum_{i=1}^{n} w^T x_i\,(\alpha_{1i} - \alpha_{2i})$$

$$X = \begin{bmatrix} \leftarrow x_1 \rightarrow \\ \leftarrow x_2 \rightarrow \\ \vdots \\ \leftarrow x_n \rightarrow \end{bmatrix}, \alpha_1 = \begin{bmatrix} \alpha_{11} \\ \alpha_{12} \\ \vdots \\ \alpha_{1n} \end{bmatrix}, \alpha_2 = \begin{bmatrix} \alpha_{21} \\ \alpha_{22} \\ \vdots \\ \alpha_{2n} \end{bmatrix}$$

Expressing the above equations in matrix and vector form (where $w = X^T(\alpha_1 - \alpha_2)$), we can get

$$\Phi(\alpha_1, \alpha_2, \beta_1, \beta_2) = -\frac{1}{2}(\alpha_1 - \alpha_2)^T XX^T(\alpha_1 - \alpha_2) - \epsilon(\alpha_1 + \alpha_2)^T \bar{1} + (\alpha_1 - \alpha_2)^T y$$

$$\begin{aligned} \text{such that, } & \alpha_{1i} + \beta_{1i} = C, \ \forall\, i = 1 \ldots n \\ & \alpha_{2i} + \beta_{2i} = C, \ \forall\, i = 1 \ldots n \\ & \alpha_{1i} \geqslant 0, \ \beta_{1i} \geqslant 0, \ \alpha_{2i} \geqslant 0, \ \beta_{2i} \geqslant 0 \\ & \alpha_{1i} \leqslant C, \alpha_{1i} \leqslant C, \alpha_{2i} \leqslant C, \beta_{1i} \leqslant C, \beta_{2i} \leqslant C \end{aligned}$$

Since the KKT multipliers $\alpha_1, \alpha_2, \beta_1, \beta_2$ are all non-negative and bounded, and also because the dual function depends on these bounded multipliers, it can't take infinite values by the virtue of the above mentioned conditions.

On the other hand, the lagrangian function took infinite values for certain cases because there was no upper bound on the multipliers $\alpha_1, \alpha_2, \beta_1, \beta_2$. Restricting the Lagrangian function to give finite values gave us the conditions describing the bound of the KKT multipliers, and thus ensuring that the dual function doesn't take infinite values.

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:
The dual form of the SVR problem can be given by,

$$\max_{\alpha_1,\alpha_2,\beta_1,\beta_2} \Phi(\alpha_1,\alpha_2,\beta_1,\beta_2)$$
$$\max_{\alpha_1,\alpha_2,\beta_1,\beta_2} -\frac{1}{2}(\alpha_1-\alpha_2)^T XX^T(\alpha_1-\alpha_2) - \epsilon(\alpha_1+\alpha_2)^T\bar{1} + (\alpha_1-\alpha_2)^T y$$

Since the above expression is dependent on the KKT multipliers α_1 and α_2 alone, we can accordingly write the dual problem based on those 2 multipliers in the following way,

$$\begin{aligned} \max_{\alpha_1,\alpha_2} \quad & -\frac{1}{2}(\alpha_1-\alpha_2)^T XX^T(\alpha_1-\alpha_2) - \epsilon(\alpha_1+\alpha_2)^T\bar{1} + (\alpha_1-\alpha_2)^T y \\ \text{such that,} \quad & \alpha_{1i} \geqslant 0, \alpha_{2i} \geqslant 0, \ \ \forall\, i = 1 \dots n \\ & \alpha_{1i} \leqslant C, \alpha_{2i} \leqslant C, \ \ \forall\, i = 1 \dots n \end{aligned}$$

The dual form of the SVR expressed as the above maximization problem can be solved using quadratic optimization problem since the above expression resembles the general form of quadratic optimization problem under constraints.

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

Solution:
The KKT conditions for the above SVR problem can be written as follows,

$$\begin{aligned} \underline{w \text{ that gives us the dual function:}} \quad & w = \sum_{i=1}^{n} (\alpha_{1i} - \alpha_{2i})\, x_i = 0 \\ \underline{\text{Relation between KKT multipliers:}} \quad & \alpha_{1i} + \beta_{1i} = C \\ & \alpha_{2i} + \beta_{2i} = C \\ \underline{\text{Complementary Slack:}} \quad & \alpha_{1i}(y_i - w^T x_i - \epsilon - \xi_{1i}) = 0 \\ & \alpha_{2i}(-y_i + w^T x_i - \epsilon - \xi_{2i}) = 0 \\ & \beta_{1i}(-\xi_{1i}) = 0 \\ & \beta_{2i}(-\xi_{2i}) = 0 \end{aligned}$$

These conditions can be re-written in terms of α_1 and α_2,

$$\underline{w \text{ that gives us the dual function:}} \quad w = \sum_{i=1}^{n} (\alpha_{1i} - \alpha_{2i})\, x_i = 0$$

$$\underline{\text{Complementary Slack:}} \quad \alpha_{1i}(y_i - w^T x_i - \epsilon - \xi_{1i}) \leqslant 0$$
$$\alpha_{2i}(-y_i + w^T x_i - \epsilon - \xi_{2i}) \leqslant 0$$
$$(C - \alpha_{1i})(-\xi_{1i}) = 0$$
$$(C - \alpha_{2i})(-\xi_{2i}) = 0$$

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.

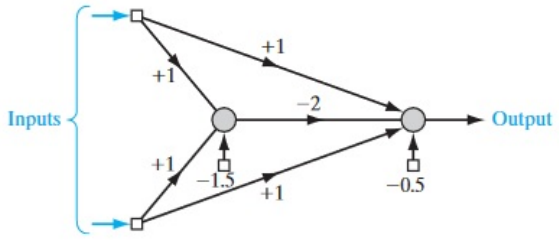


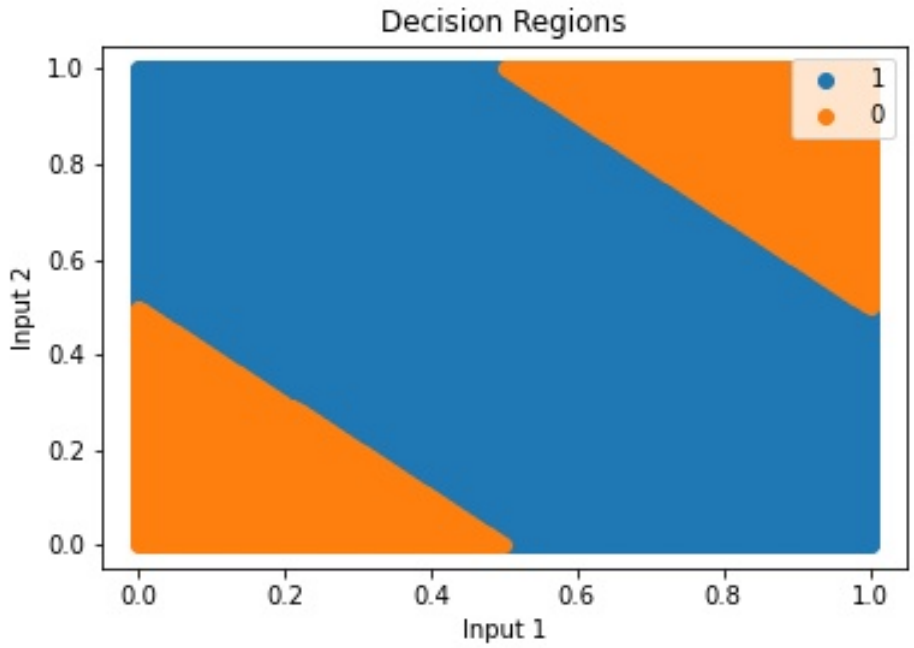
Figure 1: ANN for XOR problem

Solution:

Truth Table:

Input 1	Input 2	Hidden Neuron (-1.5)	Output (-0.5)	XOR
0	0	0 (0)	0 (0)	0
0	1	0 (1)	1 (1)	1
1	0	0 (1)	1 (1)	1
1	1	1 (2)	0 (0)	0

Decision Region:



(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:

We can write the backpropagation algorithm for the Feed-forward neural network in 4 steps that are,

$$\delta^L = \nabla_a C * \sigma'(z^L) \tag{1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) * \sigma'(z^l) \tag{2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{4}$$

where C is the squared loss function, δ^L is the partial derivative of the loss function with respect to the final hidden layer (layer L) weights (w^L_{jk} and b^L_j). δ^l gives the partial derivatives of loss function with respect to layer l's weights (w^l_{jk} and b^l_j).

When the weights are all zero, that is the origin of the weight space, we get the RHS of equation (2) to be zero. This results in the equations (3), (3) and (4) also being zero, by the

virtue of equation (2) being zero. Thus we get the partial derivates of the loss function with respect to the weights and bias parameters to be zero, which is a stationary point. This result is irrespective of the activation function used in the Feed-forward network. This proves that the origin in weight space is a stationary point of the error function.

(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:
Vanishing Gradients:

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad \sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2} \quad \sigma'(0) = \frac{1}{4} = 0.25$$

Sigmoid Function causes gradients to vanish, whenever the value is either too high or too low, which happens when the weights are initialised with high magnitude values. This problem can also occur in a deep neural network as the gradients compound exponentially, therefore even the maximum value of 0.25 also can eventually vanish.

Non Zero-Centered:
If a sigmoid function is not zero-centered, the gradient updates are likely to be either completely positive or completely negative. Therefore, the weights move in the same direction and thereby gradient descent takes a zig-zag path rather than the shortest path to react optimum. This decreases the efficiency of training, as it takes much longer time to converge at the optimum.

(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:
When all the weights are initialized to the same value, every neuron effectively calculates the same $\sigma(w^\top x + b)$ making it redundant. Therefore, the network won't be able to utilize

all nodes to capture patterns.

Biases are commonly initialised to 0, as the asymmetry can be provided by the weights. It also helps in tackling vanishing gradients problem, as the maximum gradient for sigmoid function occurs at 0. Weights are randomly initialised, generally a random number is sampled from normal distribution. However, recent literature also suggest various ways to improve initialisation using size of layers such as He initialisation, $W[l] =$ (Random sampling by standard normal) $*\sqrt{2/(size[l-1])}$ where $W[l]$ is the weight of the l^{th} layer.

(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

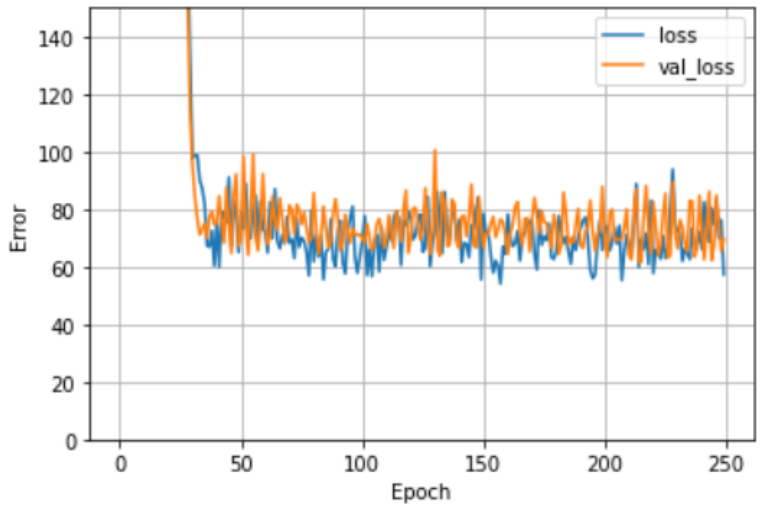
$$y = 233.846 * (1 - exp(-0.006042x)) + \epsilon$$

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.

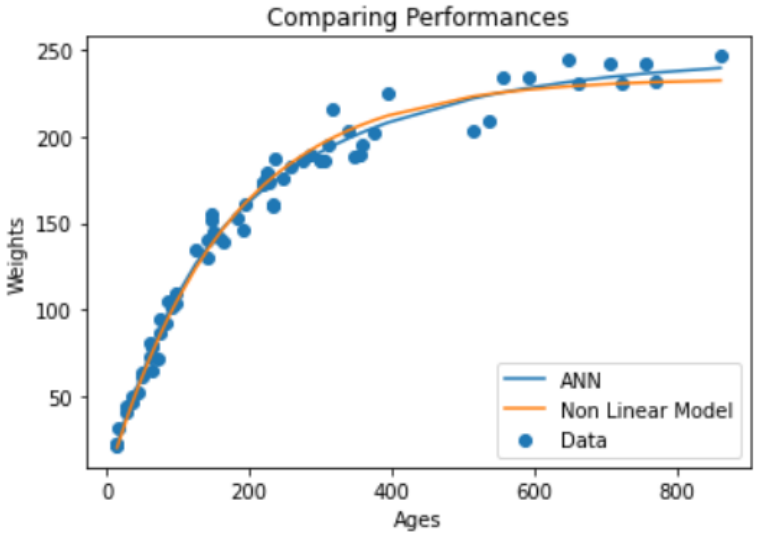
Ages (days)	Weights (mg)	Ages (days)	Weights (mg)	Ages (days)	Weights (mg)	Ages (days)	Weights (mg)
15	21.66	75	94.6	218	174.18	338	203.23
15	22.75	82	92.5	218	173.03	347	188.38
15	22.3	85	105	219	173.54	354	189.7
18	31.25	91	101.7	224	178.86	357	195.31
28	44.79	91	102.9	225	177.68	375	202.63
29	40.55	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:
The designed FFN initially was underfitting with a single hidden layer, adding another hidden layer decreased the bias. However, this led to overfitting, which was evident by plotting the validation loss. This issue was later fixed by adding a dropout layer.
Loss Plot:



The ANN improved the mean squared error to 60.682, compared to the mean squared error of the given model which was 76.116.



Colab Notebook for ANN model

(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: Colab Notebook for Backpropagation Algorithm

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

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

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

Solution:

Pseudo code for Adaboost

$w^1 = 1$

for $t = 1 \dots T$ **do**

$h_t = WeakLearner(S, w^t)$

$\gamma_t = \frac{1}{2} - \frac{1}{\sum_i w_i^t} \sum_{i=1}^{m} w_i^t \frac{|h_t(x_i) - y_i|}{2}$

$\beta_t = \frac{(0.5+\gamma_t)}{(0.5-\gamma_t)}$

$l_{t,i} = |h_t(x_i) - y_i|/2$

$w_i^{t+1} = w_i^t \beta_{l_{t,i}}^t$

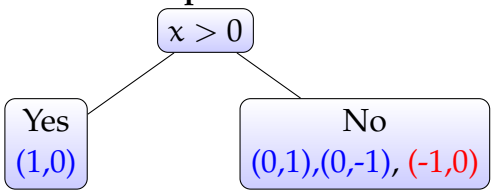
end for

$h(x) = \text{sign}(\sum_{t=1}^{T} (\log(\beta_t) h_t(x))$

The weight vector $w^{[t]} = [w_1^{[t]}, w_2^{[t]}, w_3^{[t]}, w_4^{[t]}]$ is formulated in the order of the data points $(1,0), (0,1), (0,-1), (-1,0)$

Iteration T = 1

Decision stump for the decision $x > 0$



3 points correctly classified, 1 point incorrectly classified $\rightarrow (-1,0)$

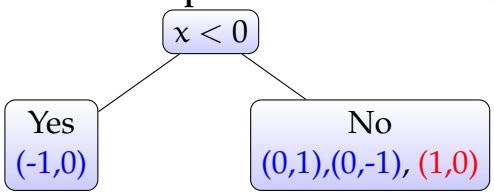
$$w^{[0]} = [1,1,1,1] \rightarrow \text{Initialized weights}$$
$$\gamma_1 = \frac{1}{2} - \frac{1}{4}[0 + 1(1) + 0 + 0] = \frac{1}{4}$$
$$\beta_1 = \frac{3/4}{1/4} = 3$$
$$l_1^1 = l_2^1 = l_3^1 = 0, l_4^1 = 1$$
$$w^{[1]} \rightarrow [1,1,1,1(\beta_1)] \rightarrow [1,1,1,3]$$

For first iteration, the decision stump is,

$$h_1(x) = \begin{cases} x > 0, & +1 \\ x < 0, & -1 \end{cases}$$

Iteration T = 2

Decision stump for the decision $x < 0$



3 points correctly classified, 1 point incorrectly classified $\rightarrow (1,0)$

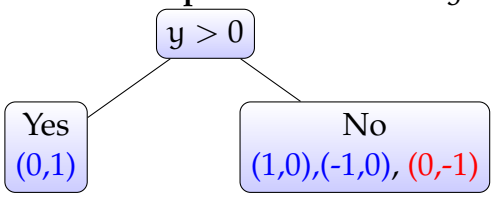
$$w^{[1]} = [1,1,1,3]$$
$$\gamma_2 = \frac{1}{2} - \frac{1}{6}[0 + 1(1) + 0 + 0] = \frac{1}{3}$$
$$\beta_2 = \frac{5/6}{1/6} = 5$$
$$l_2^2 = l_3^2 = l_4^2 = 0, l_1^2 = 1$$
$$w^{[2]} \rightarrow [1(\beta_2),1,1,3] \rightarrow [5,1,1,3]$$

For second iteration, the decision stump is,

$$h_2(x) = \begin{cases} x < 0, & +1 \\ x > 0, & -1 \end{cases}$$

Iteration T = 3

Decision stump for the decision $y > 0$



3 points correctly classified, 1 point incorrectly classified $\rightarrow (0, -1)$

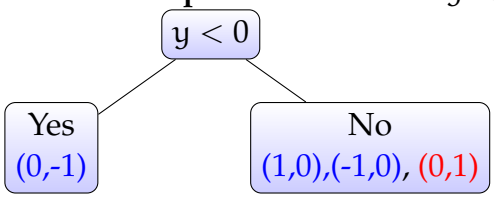
$$w^{[2]} = [5, 1, 1, 3]$$
$$\gamma_3 = \frac{1}{2} - \frac{1}{10}[0 + 1(1) + 0 + 0] = \frac{2}{5}$$
$$\beta_3 = \frac{9/10}{1/10} = 9$$
$$l_1^3 = l_2^3 = l_4^3 = 0, l_3^3 = 1$$
$$w^{[3]} \rightarrow [5, 1, 1(\beta_3), 3] \rightarrow [5, 1, 9, 3]$$

For third iteration, the decision stump is,,

$$h_3(x) = \begin{cases} y > 0, & -1 \\ y < 0, & +1 \end{cases}$$

Iteration T = 4

Decision stump for the decision $y < 0$



3 points correctly classified, 1 point incorrectly classified $\rightarrow (0, 1)$

$$w^{[3]} = [5, 1, 9, 3]$$
$$\gamma_4 = \frac{1}{2} - \frac{1}{18}[0 + 1(1) + 0 + 0] = \frac{4}{9}$$
$$\beta_4 = \frac{17/18}{1/18} = 17$$
$$l_1^4 = l_3^4 = l_4^4 = 0, l_2^4 = 1$$
$$w^{[4]} \rightarrow [5, 1(\beta_4), 19, 3] \rightarrow [5, 17, 9, 3]$$

For fourth iteration, the decision stump is,

$$h_4(x) = \begin{cases} y < 0, & -1 \\ y > 0, & +1 \end{cases}$$

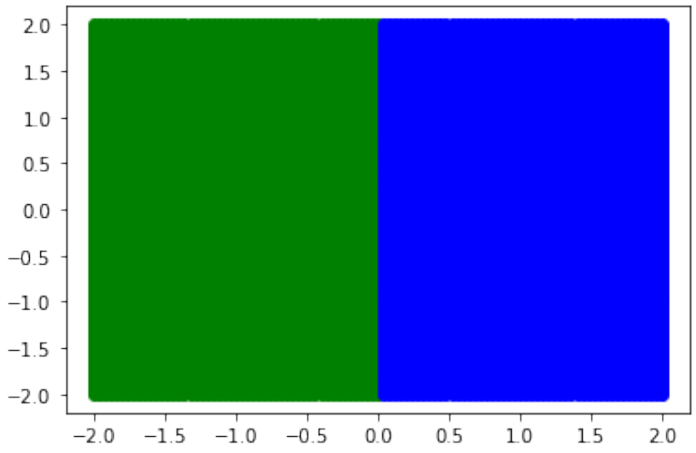
ii. (2 points) Draw the classifier at each timestep.

Solution:

Color	Label
Blue	(+) $\rightarrow$ (+1)
Green	(-) $\rightarrow$ (−1)

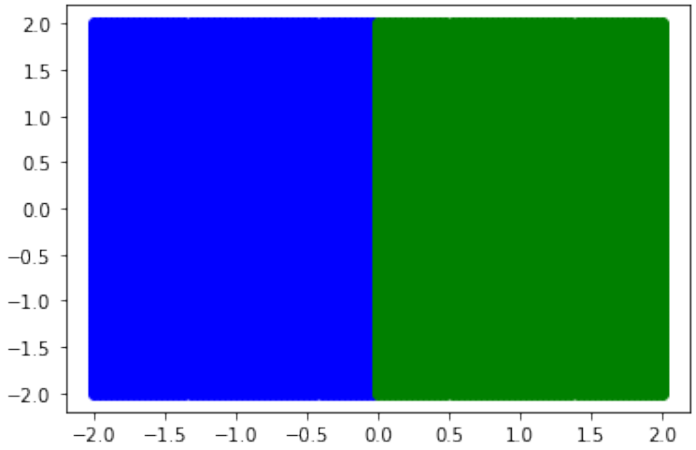
Classfier at the end of first timestep

$$h(x) = sign(log(\beta_1)h_1(x))$$
$$= h_1(x)$$



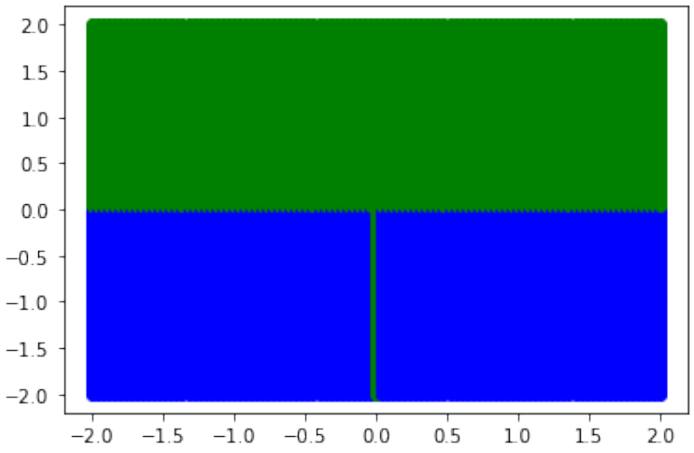
Classfier at the end of second timestep

$$\begin{aligned} h(x) &= \text{sign}(\log(\beta_1)h_1(x) + \log(\beta_2)h_2(x)) \\ &= \text{sign}(\log(3)h_1(x) + \log(5)h_2(x)) \end{aligned}$$



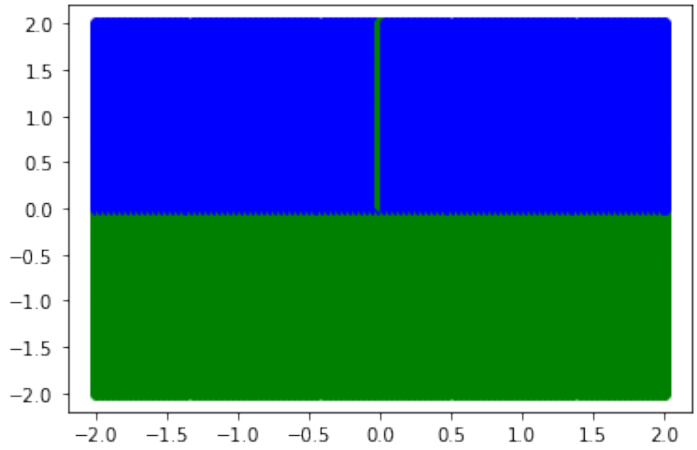
Classfier at the end of third timestep

$$\begin{aligned} h(x) &= \text{sign}(\log(\beta_1)h_1(x) + \log(\beta_2)h_2(x) + \log(\beta_3)h_3(x)) \\ &= \text{sign}(\log(3)h_1(x) + \log(5)h_2(x) + \log(9)h_3(x)) \end{aligned}$$



Classfier at the end of fourth timestep

$$\begin{aligned} h(x) &= \text{sign}(\log(\beta_1)h_1(x) + \log(\beta_2)h_2(x) + \log(\beta_3)h_3(x) + \log(\beta_4)h_4(x)) \\ &= \text{sign}(\log(3)h_1(x) + \log(5)h_2(x) + \log(9)h_3(x) + \log(17)h_4(x)) \end{aligned}$$



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

Solution:
Decision stumps are very weak learners. This is because they can only make a decision based on a single feature, called as a (1-level) decision. Moreover, it makes a decision such that it splits the dataset into 2 halves, while trying to acheive the maximum accuracy in doing so. General datasets are not always linearly separable and hence, decision stumps are weak learners.

The training error for the 4 time steps taken in the order $(1,0),(0,1),(0,-1),(-1,0)$ are given by,

T = 1

$$\begin{aligned} h(x) &= \text{sign}(\log(\beta_1)h_1(x)) \\ &= \text{sign}(\log(3)h_1(x)) = h_1(x) \\ \text{Training Error} &= \frac{1}{4}(0+0+0+1) \\ &= \frac{1}{4} \end{aligned}$$

T = 2

$$\begin{aligned} h(x) &= \text{sign}(\log(\beta_1)h_1(x) + \log(\beta_2)h_2(x)) \\ &= \text{sign}(\log(3)h_1(x) + \log(5)h_2(x)) \\ \text{Training Error} &= \frac{1}{4}(1+0+0+0) \\ &= \frac{1}{4} \end{aligned}$$

T = 3

$$\begin{aligned} h(x) &= \text{sign}(\log(\beta_1)h_1(x) + \log(\beta_2)h_2(x) \\ &\quad + \log(\beta_3)h_3(x)) \\ &= \text{sign}(\log(3)h_1(x) + \log(5)h_2(x) \\ &\quad + \log(9)h_3(x)) \\ \text{Training Error} &= \frac{1}{4}(0+0+1+0)) \\ &= \frac{1}{4} \end{aligned}$$

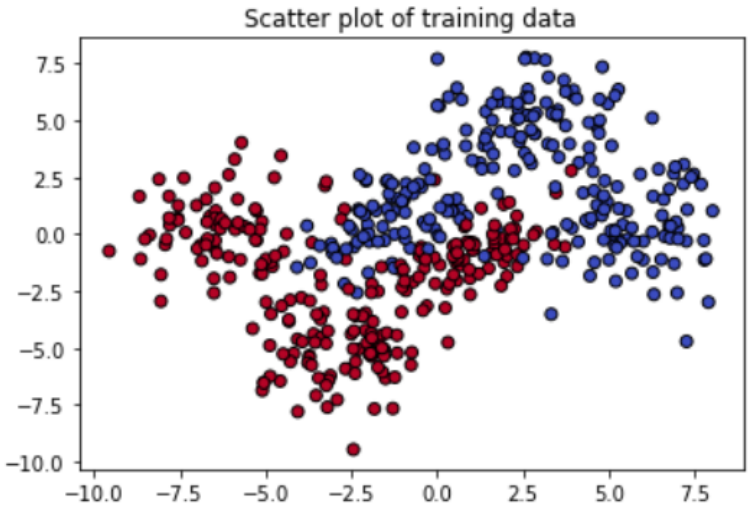
T = 4

$$\begin{aligned} h(x) &= \text{sign}(\log(\beta_1)h_1(x) + \log(\beta_2)h_2(x) \\ &\quad + \log(\beta_3)h_3(x) + \log(\beta_4)h_4(x)) \\ &= \text{sign}(\log(3)h_1(x) + \log(5)h_2(x) \\ &\quad + \log(9)h_3(x) + \log(17)h_4(x)) \\ \text{Training Error} &= \frac{1}{4}(0+0+0+0) \\ &= 0 \end{aligned}$$

(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.**)

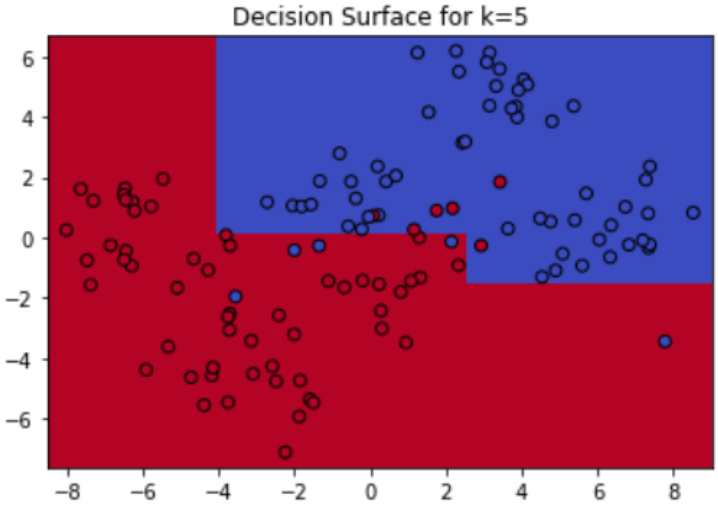
i. (2 points) Plot the training data.

Solution:



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

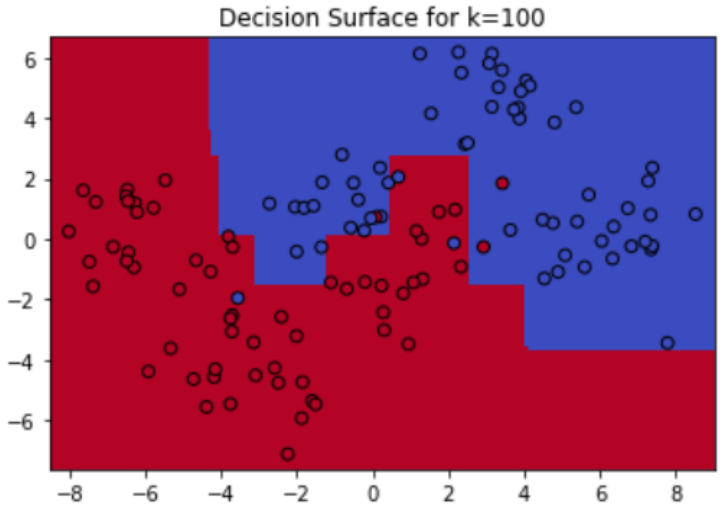
Solution:



Accuracy of the predictions on the test set with k = 5 is 90.833%

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

Solution:



Accuracy of the predictions on the test set with $k = 100$ is 95%

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: Colab Notebook for AdaBoost model

(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

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

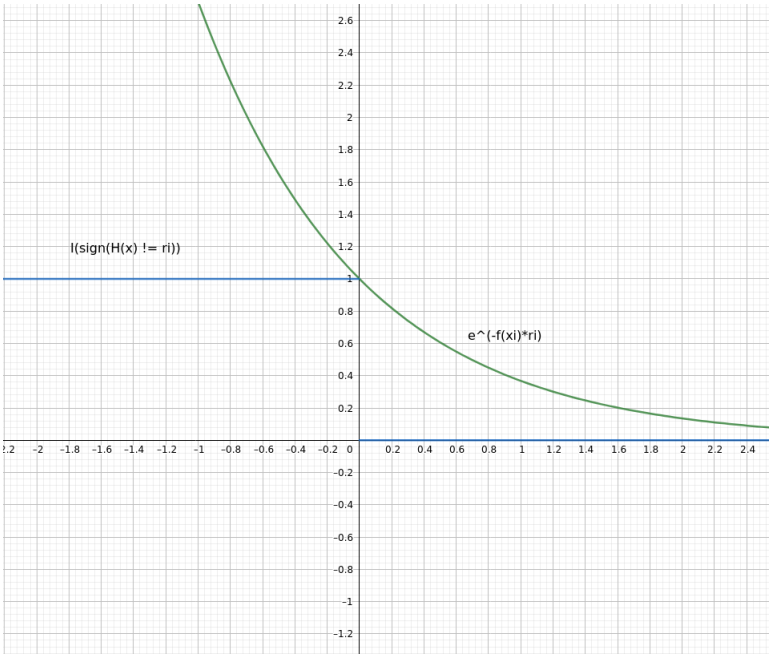
Show that the training error of the final classifier can be bounded from above by an exponential loss function :

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

where I is the indicator function.

Solution:

Consider the functions $I(H(x_i) \neq r_i)$ and $e^{-f(x_i) r_i}$. When we plot these functions with respect to $f(x_i)$, we get



This clearly tells us that the function $I(H(x_i) \neq r_i)$ is upper bounded by the function $e^{-f(x_i)r_i}$ for all data points $i = 1 \dots n$.

$$\implies I(H(x_i) \neq r_i) \leqslant e^{-f(x_i)r_i}$$
$$\implies I(sign(f(x_i)) \neq r_i) \leqslant e^{-f(x_i)r_i}$$

We use the property that addition of multiple inequalities in the same direction preserves the inequality. We can extend this property to 'm' data points and state that,

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

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 $\{\lambda_i := \sigma_i^2\}_{i=1,\dots,N}$ are called the singular values $\{\sigma_i\}_{i=1,\dots,N}$ of X.)

Solution:
Given $X \in \mathbb{R}^{N\times D}$ where $D >> N$.

Thus, we can say that XX^T and X^TX are square matrices even if X is not. And we have that $XX^T \in \mathbb{R}^{N\times N}$ and $X^TX \in \mathbb{R}^{D\times D}$.

(i) Let the matrix XX^T have L distinct non-zero eigenvalues as $\{\alpha_i\}_{i=1,\dots,L}$ and corresponding eigenvectors as $\{u_i\}_{i=1,\dots,L}$. From this we can write that,

$$XX^Tu_i = \alpha_i u_i, \quad \forall i = 1 \dots L$$

Further, we can use the inner product properties to prove this,

$$\langle XX^Tu_i, u_i\rangle = \langle X^Tu_i, X^Tu_i\rangle$$

Also,

$$\langle XX^Tu_i, u_i\rangle = \alpha_i\langle u_i, u_i\rangle$$

Finally we get that,

$$\langle X^Tu_i, X^Tu_i\rangle = \|X^Tu_i\|^2 = \alpha_i\langle u_i, u_i\rangle = \alpha_i\|u_i\|^2$$

$$\|X^Tu_i\|^2 = \alpha_i\|u_i\|^2 > 0$$

Hence, by the property of norms we can say that $\alpha_i > 0$ (because we considered $\alpha_i \neq 0$) and hence the non-zero eigenvalues of XX^T are all **positive**. Moreover, we can say that L = N because of the fact that XX^T is positive definite. This is because,

$$\langle XX^Tv, v\rangle = \langle X^Tv, X^Tv\rangle = \|X^Tv\|^2$$

From the question, we know that rows of X are linearly independent and hence we can say that columns of X^T are linearly independent, which proves that the rank of matrix X^T is N.

Hence the null space of X^T is empty.
So, we can say that,

$$\langle XX^T v, v \rangle = ||X^T v||^2 > 0$$

which means that XX^T is positive-definite and has N positive eigen-values. Hence, $L = N$

(ii) From the previous argument, considering the same set of eigenvalues and eigenvectors,

$$XX^T u_i = \alpha_i u_i, \quad \alpha_i \neq 0, u_i \neq [0, \ldots, 0]^T$$

Consider,

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

Comparing the first and last terms of the above set of equations, we get,

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

This proves that the non-zero eigenvalues of $X^T X$ are also α_i $(\forall\ i = 1, ..., N)$

(iii) Let $AM(\lambda)_B$ denote the algebraic multiplicity of the eigenvalue λ for the matrix B.
We have that $AM(\alpha_i)_{XX^T} = AM(\alpha_i)_{X^T X}$ for $\alpha_i \neq 0$ and $AM(0)_{XX^T} = 0$. Since, the eigen-values of $X^T X$ that are not equal to the eigenvalues of XX^T are zeroes, we can say that $AM(0)_{X^T X} = (D - N)$

(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 $X^T X$. 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 $X^T X$; 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: Following the same argument from the previous question, considering the set of eigenvectors $\{u_i\}_{i-=1,\ldots,N}$ of XX^T to be an orthonormal set. Similarly we can consider $\{v_j\}_{j=1,\ldots,N}$ for $X^T X$ to be an orthonormal set, because of the corresponding eigenvalues $\{\alpha_i\}_{i=1,\ldots,N}$ they have in common with the matrix XX^T.

$$XX^T u_i = \alpha_i u_i, \quad \alpha_i \neq 0, u_i \neq [0, \ldots, 0]^T, i = 1, \ldots, N$$
$$X^T X v_i = \alpha_i v_i, \quad \alpha_i \neq 0, v_i \neq [0, \ldots, 0]^T, i = 1, \ldots, N$$

Consider,

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

Comparing the first and last terms of the above set of equations, we get,

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

where,

$$X^T u_i, \;\; i = 1, \ldots, N$$

are eigenvectors for the matrix $X^T X$ but not an orthonomoral basis of eigenvectors. Since, they are already orthogonal we can make the basis of eigenvectors orthonormal by setting unit norm.

$$v_i = \frac{X^T u_i}{\|X^T u_i\|} = \frac{X^T u_i}{\sqrt{\alpha_i}} = \frac{X^T u_i}{\sigma_i}$$

This gives us an easier way of computing the orthonormal set of eigenvectors $\{v_j\}_{j=1,\ldots,N}$ corresponding to the non-zero eigenvalues $\{\alpha_j\}_{j=1,\ldots,N}$ without eigenvalue decomposition of XX^T. We also get the singular values of the matrix X as the square root of the eigenvalues α_i as $\sigma_i = \sqrt{\alpha_i}$

(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:
Since the matrix $S = \frac{1}{N} X^T X \in \mathbb{R}^{D \times D}$, we have the time complexity of finding the eigenvectors of S (i.e.) $\{v_j\}_{j=1,\ldots,D}$, to be $O(D^3)$ which is very high, given that $D >> N$. From the answers to the previous questions (a) and (b) it is clear that the computational time can be reduced by finding the orthonormal eigenvectors of the matrix $S' = \frac{1}{N} XX^T \in \mathbb{R}^{N \times N}$ instead, because the time complexity in this case is $O(N^3)$.

Furthermore, to derive the orthonormal set $\{v_j\}_{j=1,\ldots,N}$ from $\{u_i\}_{i=1,\ldots,N}$ we can use the property from previous question (b) which is equal to $v_i = \frac{X^T u_i}{\sigma_i}$. Computing these orthonormal set of eigenvectors takes an additional time complexity of $O(DN^2)$ by virtue of the computations involved below, (i.e.)

1. $X^T \in \mathbb{R}^{D \times N}$ and $u_i \in \mathbb{R}^N$. So, computing each $v_i = X^T u_i$ has a time complexity of $O(DN)$

2. Computing the N eigenvectors $\{v_j\}_{j=1,\ldots,N}$ introduces N additional computations and hence the effective time complexity becomes $O(DN^2)$

3. This gives us the orthonormal set of eigenvectors $\{v_j\}_{j=1,\ldots,N}$ for the matrix S without doing the eigenvalue decomposition of S.

4. The total time complexity is $O(N^3)+O(DN^2)$ and it is efficient than the operation that gives time complexity of $O(D^3)$

(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.)

Solution:
The principal directions are given by the eigenvectors of the data covariance matrix S, and are orthonormal to each other.

$$u_i^T u_j = \begin{cases} 1, & i = j \\ \\ 0, & i \neq j \end{cases}$$

This necessitates that,

$$\hat{U}^T\hat{U} = I$$

But, any general solution $\hat{U}$ to the below equation should satisfy the conditions,

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

Conditions :

$$\hat{U}^T\hat{U} = I$$
$$S\hat{U} = \hat{U}H$$

Specific Solution :

$$\hat{U}= \begin{bmatrix} \cdots v_{M+1}^T \cdots \\ \cdots v_{M+2}^T \cdots \\ \vdots \\ \cdots v_D^T \cdots \end{bmatrix}, S\hat{U}= \begin{bmatrix} \vdots & \vdots & \cdots & \vdots \\ S\hat{v}_{M+1} & S\hat{v}_{M+2} & \cdots & S\hat{v}_D \\ \vdots & \vdots & \cdots & \vdots \end{bmatrix}$$

$\hat{U}$ is the solution matrix formed by the (D-M) orthonormal eigenvectors $\{v_j\}_{j=M+1,\ldots,D}$ of S with as it's columns and $\{\alpha_j\}_{j=M+1,\ldots,D}$ as the corresponding eigenvalues. Using the matrices defined above we get,

$$\hat{U}^TS\hat{U}= \begin{bmatrix} \alpha_{M+1} & 0 & \cdots & 0 \\ 0 & \alpha_{M+2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \alpha_D \end{bmatrix}$$

Also, subsituting $\hat{U}^T\hat{U} = I$ and $S\hat{U} = \hat{U}H$ in the expression for $\tilde{J}$ gives us,

$$\tilde{J} = Tr\left\{\hat{U}^T\hat{U}H\right\} + Tr\{H(I-I)\} = Tr\{H\}$$

Since, we get $\hat{U}^TS\hat{U} = H$, we can write H as a diagonal matrix, which is,

$$H= \begin{bmatrix} \alpha_{M+1} & 0 & \cdots & 0 \\ 0 & \alpha_{M+2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \alpha_D \end{bmatrix}$$

Thus we get,

$$\min\left(\tilde{J}\right) = Tr\{H\} = \sum_{i=M+1}^{D} \alpha_i$$

where $\{\alpha_j\}_{j=M+1,\ldots,D}$ are the lowest (D-M) eigenvalues of the matrix S and $\{v_j\}_{j=M+1,\ldots,D}$ are the eigenvectors corresponding to these minimum eigenvalues.

If $\hat{U}$ is considered to be a general solution to the above problem that minimizes $\tilde{J}$ for an

appropriate H matrix, then we have,

$$\begin{aligned} S\widehat{U} &= \widehat{U}H \\ \widehat{U}^T S\widehat{U} &= \widehat{U}^T\widehat{U}H \\ \text{We get, } H &= \widehat{U}^T S\widehat{U} \\ H^T &= \widehat{U}^T S^T\widehat{U} \end{aligned}$$

Since the covariance matrix S is symmetric (i.e.) $S^T = S$, we also get $H^T = H$ and hence, H is symmetric for a general solution to $\widetilde{J}$. Thus, using the symmetric property of H we can diagonalize it using an orthogonal matrix Y, to relate the general solution to the specific solution we already know. We can now write,

$$H = Y^T D Y \ \ (\text{where } Y^{-1} = Y^T)$$

Since, $Y^T, Y, D \in \mathbb{R}^{(D-M)\times(D-M)}$ we also get $H \in \mathbb{R}^{(D-M)\times(D-M)}$

$$\begin{aligned} S\widehat{U} &= \widehat{U}Y^T DY \\ S(\widehat{U}Y^T) &= \widehat{U}Y^T DYY^T = (\widehat{U}Y^T)D \\ \text{We get, } SU' &= U'D \end{aligned}$$

where $U' = \widehat{U}Y^T$ and the columns of U' are the eigenvectors of S corresponding to the eigenvalues that are the diagonal entries of the diagonal matrix D. Since, H and D have the same (D-M) eigenvalues, by the virtue of diagonalization, we can get the eigenvalues of D as $\{\alpha_j\}_{j=M+1,\ldots,D}$ (where these are the minimum (D-M) eigenvalues of the matrix S.

Also, because the minimum value of $\widetilde{J} = Tr\{H\}$, we get,

$$\widetilde{J}_{min} = \sum_{i=M+1}^{D} \alpha_i$$

We converted the general problem to be in the form of the specific solution, where we have $H = D$ and $U' = \widehat{U}Y^T$ but the minimum value for $\widetilde{J}$ remains the same.