Introduction to ML (CS771), 2024-2025-Sem-I **Midsem exam. September 15, 2024**	**Total Marks**	**60**
	Duration	**2 hours**
Name	**Roll No.**	

Instructions:

1.	Clearly write your name (in block letters) and roll number in the provided boxes above.
2.	Write your final answers concisely in the provided space. You may use blue/black pen.
3.	We may not be able to provide clarifications during the exam. If any aspect of some question appears ambiguous/unclear to you, please state your assumption(s) and answer accordingly.
4.	The last page (page 6) of this booklet can be used for rough work.

Section 1 (Short Answer Questions): Answer the following questions concisely in the space provided below the question.

1.1 Assume a scalar-valued function $f(\boldsymbol{w})$, with $\boldsymbol{w} = [w_1, w_2, w_3, w_4]^\top \in \mathbb{R}^4$, defined as $f(\boldsymbol{w}) = (w_1 - w_2)^2 + (w_2 - w_3)^2 + (w_3 - w_4)^2 + (w_4 - w_1)^2$. Write down the final expression (don't show the steps) of gradient of $f(\boldsymbol{w})$ w.r.t. $\boldsymbol{w}$. Without using the gradient, can you answer what is the minima of this function? If so, answer what it is. If not, state why you can't. Also, is $f(\boldsymbol{w})$ convex? Briefly justify the answer. **(2+2+2 =6 marks)**

$$\nabla_{\boldsymbol{w}} f(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{w})}{\partial w_1} \\ \frac{\partial f(\boldsymbol{w})}{\partial w_2} \\ \frac{\partial f(\boldsymbol{w})}{\partial w_3} \\ \frac{\partial f(\boldsymbol{w})}{\partial w_4} \end{bmatrix} = \begin{bmatrix} 2(w_1 - w_2) - 2(w_4 - w_1) \\ -2(w_1 - w_2) + 2(w_2 - w_3) \\ -2(w_2 - w_3) + 2(w_3 - w_4) \\ -2(w_3 - w_4) + 2(w_4 - w_1) \end{bmatrix} = \begin{bmatrix} 4w_1 - 2w_2 - 2w_4 \\ 4w_2 - 2w_1 - 2w_3 \\ 4w_3 - 2w_2 - 2w_4 \\ 4w_4 - 2w_1 - 2w_3 \end{bmatrix}$$

Since the function only contains non-negative terms, the smallest value it can take is 0 and is attained when $w_1 = w_2 = w_3 = w_4$, so any vector $\boldsymbol{w}$ with all its entries being equal will be a minima. Since there can be multiple such vectors, the function has multiple minima and therefore it cannot be (strictly) convex but it is still convex (Hessian is PSD).

1.2 Which of these are examples of convex sets? (1) the set of D-dimensional vectors with all non-negative entries, (2) set of D-dimensional vectors that have at most $K < D$ nonzero entries. Briefly justify your answers. **(2+2 =4 marks)**

(1) The set of vectors with all non-negative entries is a convex set since any convex combination of two vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ from this set, i.e., $\boldsymbol{x}_3 = \lambda \boldsymbol{x}_1 + (1 - \lambda)\boldsymbol{x}_2$ will also be a vector with all non-negative entries.

(2) The set of vectors with at most K is not convex since any convex combination of two vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ from this set, i.e., $\boldsymbol{x}_3 = \lambda \boldsymbol{x}_1 + (1 - \lambda)\boldsymbol{x}_2$ could result in a vector with possibly more than K non-zero entries.

1.3 Given the K real-valued logits $z_1, z_2, \dots, z_K$, the softmax function produces a vector of probabilities with each probability of the form $\mu_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}$. Write down the expressions of derivative of μ_i w.r.t. z_i and also the derivative of μ_i w.r.t. $z_j, j \neq i$. **(3+3 =6 marks)**

Using the derivative rule for quotients and the chain rule of derivatives,

$$\frac{\partial \mu_i}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}\right) = \frac{\left(\sum_{j=1}^{K} \exp(z_j)\right) \times \exp(z_i) - \exp(z_i) \times \exp(z_i)}{\left(\sum_{j=1}^{K} \exp(z_j)\right)^2} = \mu_i(1-\mu_i)$$

$$\frac{\partial \mu_i}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}\right) = \frac{\left(\sum_{j=1}^{K} \exp(z_j)\right) \times 0 - \exp(z_i) \times \exp(z_j)}{\left(\sum_{j=1}^{K} \exp(z_j)\right)^2} = -\mu_i\mu_j$$

1.4 Assume two scalar inputs x and z. The RBF kernel based similarity between these two inputs is defined as $k(x,z) = \exp(-\gamma(x-z)^2)$. Assume $\gamma = 1$ for simplicity. Show that $k(x,z) = \phi(x)^\top \phi(z)$ and write down the expression of the feature mapping ϕ. You may use the series expansion result $\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ **(6 marks)**

$k(x,z) = \exp(-(x-z)^2) = \exp(-x^2 - z^2 + 2xz) = \exp(-x^2)\exp(-z^2)\exp(2xz)$.

We can expand $\exp(2xz)$ using the infinite series expansion

$$\exp(2xz) = \sum_{n=0}^{\infty} \frac{(2xz)^n}{n!} = \sum_{n=0}^{\infty} \frac{2^n x^n z^n}{n!}$$

Therefore $k(x,z) = \exp(-x^2)\exp(-z^2)\sum_{n=0}^{\infty} \frac{2^n x^n z^n}{n!}$

Note that we can view the above as an inner product, i.e., $k(x,z) = \phi(x)^\top \phi(z)$ where ϕ is defined a vector consisting of infinite many entries. Ignoring terms that are constants $(2^n, n!)$ and don't depend on the input, we can write ϕ as

$$\phi(x) = [\exp(-x^2)\, x^0, \exp(-x^2)\, x^1, \exp(-x^2)\, x^2, \exp(-x^2)\, x^3, \dots]$$

An identical expression can be defined for $\phi(z)$ as well with x replaced by z.

This shows that using the RBF kernel to measure similarity between two inputs is equivalent to mapping them to an infinite dimensional space and computing inner product based similarity in that space.

1.5 Suppose we want to train a Perceptron algorithm for binary classification but in a manner that the hyperplane has a margin, and not just equal margin γ on both sides but uneven margins (γ_+ towards the positive side and γ_- on the negative side; assume $\gamma_+ > \gamma_-$). Briefly describe using the necessary equation(s) how you would accomplish this. In what cases, having an uneven margin may be desirable? **(6 marks)**

The standard Perceptron uses the mistake driven update

$$\begin{aligned}&\text{if } y_n \boldsymbol{w}^\top \boldsymbol{x}_n \leq 0\\ &\qquad \boldsymbol{w}_{t+1} = \boldsymbol{w}_t + y_n \boldsymbol{x}_n\end{aligned}$$

The above doesn't ensure any margin around the hyperplane. To have a margin of $\gamma > 0$ on either side of the hyperplane, we can change the above to

$$\begin{aligned}&\text{if } y_n \boldsymbol{w}^\top \boldsymbol{x}_n \leq \gamma\\ &\qquad \boldsymbol{w}_{t+1} = \boldsymbol{w}_t + y_n \boldsymbol{x}_n\end{aligned}$$

To have an uneven margin, we can make the above mistake driven update depend on the label y_n, so we will ensure larger margin on the positive side. Then the update becomes

$$\begin{aligned}&\text{if } y_n \boldsymbol{w}^\top \boldsymbol{x}_n \leq \gamma_{y_n}\\ &\qquad \boldsymbol{w}_{t+1} = \boldsymbol{w}_t + y_n \boldsymbol{x}_n\end{aligned}$$

Uneven margin may be desirable if we want to be more cautious against erring on one of the classes, e.g., if we have imbalanced classes (say very few positive examples and we don't want to make mistakes on these rare examples).

1.6 Write down the expression for the gradient descent (GD) update for the ridge regression model which is the same as least squares (LS) linear regression with an additional ℓ_2-squared regularization on the weight vector $\boldsymbol{w}$. How is this GD update of the weight vector different from GD update for the standard LS linear regression, and what effect does it have on the weight vector for ridge regression. **(6 marks)**

For ridge regression with objective $\sum_{n=1}^{N}(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda \boldsymbol{w}^\top \boldsymbol{w}$, the gradient will be

$$\boldsymbol{g} = -2 \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\boldsymbol{x}_n + 2\lambda \boldsymbol{w}$$

Therefore, the GD update will be $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$. After substituting the gradient

$$\begin{aligned}\boldsymbol{w}^{(t+1)} &= \boldsymbol{w}^{(t)} - \eta_t \left[-2 \sum_{n=1}^{N} \left(y_n - {\boldsymbol{w}^{(t)}}^\top \boldsymbol{x}_n\right) \boldsymbol{x}_n + 2\lambda \boldsymbol{w}^{(t)}\right]\\ &= (1 - 2\lambda\eta_t)\boldsymbol{w}^{(t)} + 2\eta_t \sum_{n=1}^{N} \left(y_n - {\boldsymbol{w}^{(t)}}^\top \boldsymbol{x}_n\right) \boldsymbol{x}_n\end{aligned}$$

Note the scaling by the constant $(1 - 2\lambda\eta_t)$ which is not present in GD updates for LS linear regression. This scaling's effect is to reduce the magnitude of the entries in $\boldsymbol{w}$, which is what we want from ridge regression.

Section 2 (Long Answer Questions): Answer the following questions concisely in the space provided below the question.

2.1	The binary cross-entropy loss used in logistic regression with weight vector $\boldsymbol{w} \in \mathbb{R}^D$ is given by $L(\boldsymbol{w}) = \sum_{n=1}^{N} -[y_n \log \sigma(w^\top \boldsymbol{x}_n) + (1 - y_n)\log(1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n))]$, where $\sigma(.)$ denotes the sigmoid function. Derive the Newton's method updates for $\boldsymbol{w}$ for optimizing this loss. **Note:** If you find it more convenient with matrix-vector notation then note that the above loss can also be written as $L(\boldsymbol{w}) = -\boldsymbol{y}^\top \log \sigma(\boldsymbol{X}\boldsymbol{w}) - (1 - \boldsymbol{y})^\top \log(1 - \sigma(\boldsymbol{X}\boldsymbol{w}))$ where $\boldsymbol{X}$ denotes the $N \times D$ feature matrix and $\boldsymbol{y}$ is the $N \times 1$ label vector, and log and sigmoid functions are applied element-wise on their vector arguments **(12 marks)**

The Newton's method update has the following form

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - {\boldsymbol{H}^{(t)}}^{-1} \boldsymbol{g}^{(t)}$$

where $\boldsymbol{g}^{(t)}$ **and** $\boldsymbol{H}^{(t)}$ denote the gradient and Hessian, respectively, at iteration $\boldsymbol{t}$

Defining $\mu_n = \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n)$, the gradient of the cross-entropy loss is (as we have seen in class)

$$\boldsymbol{g} = \sum_{n=1}^{N} -(y_n - \mu_n)\boldsymbol{x}_n$$

Hessian, the second derivative, is basically the gradient's gradient w.r.t. $\boldsymbol{w}$. Thus

$$\boldsymbol{H} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{w}} = \frac{\partial}{\partial \boldsymbol{w}} \sum_{n=1}^{N} -(y_n - \mu_n)\boldsymbol{x}_n = \sum_{n=1}^{N} \frac{\partial}{\partial \boldsymbol{w}} \mu_n \boldsymbol{x}_n$$

Each term in the above summation is the gradient of a $D \times 1$ <u>vector</u> $\mu_n \boldsymbol{x}_n$ w.r.t. another $D \times 1$ vector $\boldsymbol{w}$. The result will be a <u>matrix</u> of size $D \times D$ and can be calculated using product rule of derivatives as

$$\frac{\partial}{\partial \boldsymbol{w}} \mu_n \boldsymbol{x}_n = \frac{\partial}{\partial \boldsymbol{w}} \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n)\boldsymbol{x}_n = \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n) \times 0 + \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n)\big(1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n)\big)\boldsymbol{x}_n \boldsymbol{x}_n^\top$$

Where we have used the fact that $\frac{\partial}{\partial w} \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n) = \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n)\big(1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n)\big)\boldsymbol{x}_n$

Therefore the Hessian is

$$\boldsymbol{H} = \sum_{n=1}^{N} \mu_n (1 - \mu_n)\, \boldsymbol{x}_n \boldsymbol{x}_n^\top$$

2.2 Given training data $(\boldsymbol{X}, \boldsymbol{y}) = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ with $y_n \in \{1,2,\dots,K\}$, show that class prototype vectors $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_K$ in the learning with prototypes (LwP) problem are the solution to the minimizing the following objective: $\underset{\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\dots,\boldsymbol{\mu}_K}{\arg\min} \sum_{n=1}^N \left\|\boldsymbol{x}_n - \boldsymbol{\mu}_{y_n}\right\|^2$.

Now consider a variant of the LwP problem where the labels $\boldsymbol{y} = \{y_n\}_{n=1}^N$ are not known. In this case, we must estimate not just the class prototype vectors $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_K$ but also the labels $\boldsymbol{y} = \{y_n\}_{n=1}^N$. For this "unsupervised" learning problem, give an ALT-OPT algorithm that alternates between estimating both sets of unknowns. Clearly write the update equations for the prototypes and labels in this ALT-OPT algorithm. **(14 marks)**

Consider the given optimization problem $\underset{\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\dots,\boldsymbol{\mu}_K}{\arg\min} \sum_{n=1}^N \left\|\boldsymbol{x}_n - \boldsymbol{\mu}_{y_n}\right\|^2$

Since the class prototype vectors $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_K$ are not coupled with each other in the above objective, we can optimize for each $\boldsymbol{\mu}_c, c = 1,2,\dots,K$ separately by solving

$$\hat{\boldsymbol{\mu}}_c = \underset{\boldsymbol{\mu}_c}{\arg\min} \sum_{n:y_n=c} \|\boldsymbol{x}_n - \boldsymbol{\mu}_c\|^2$$

Taking derivative of $\sum_{n:y_n=c} \|\boldsymbol{x}_n - \boldsymbol{\mu}_c\|^2$ (a quadratic function of $\boldsymbol{\mu}_c$) w.r.t. $\boldsymbol{\mu}_c$ is easy and then using first-order optimality gives the following solution for the prototype of the c^{th} class: $\hat{\boldsymbol{\mu}}_c = \frac{1}{N_c} \sum_{n:y_n=c} \boldsymbol{x}_n$ which is nothing but the LwP solution!

When the labels $\boldsymbol{y} = \{y_n\}_{n=1}^N$ are not known, we have to jointly optimize for both $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_K$ but also the labels $\boldsymbol{y} = \{y_n\}_{n=1}^N$ by solving the following optimization problem

$$\underset{\substack{\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\dots,\boldsymbol{\mu}_K \\ y_1, y_2, \dots y_N}}{\arg\min} \sum_{n=1}^N \left\|\boldsymbol{x}_n - \boldsymbol{\mu}_{y_n}\right\|^2$$

This however is not easy to solve because we have two sets of unknowns here (set 1 being $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_K$ and set 2 being $\boldsymbol{y} = \{y_n\}_{n=1}^N$).

However, ALT-OPT can rescue us, where we will solve for set 1 variables using the current best estimates of variables in set 2, and likewise solve for set 2 variables using the current best estimates of set 1 variables. For the given problem, these two steps are straightforward. :-) Basically, if someone gave us good "guesses" as estimates for the labels $\boldsymbol{y} = \{y_n\}_{n=1}^N$, we can pretend them to be the "true" labels and then solving for the class prototype vector becomes straightforward (an LwP problem) with the solution of each class prototype vector given by $\hat{\boldsymbol{\mu}}_c = \frac{1}{N_c} \sum_{n:y_n=c} \boldsymbol{x}_n$. Likewise, given these current estimates of the class prototypes $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_K$, solving for the other set of unknowns $\boldsymbol{y} = \{y_n\}_{n=1}^N$ is straightforward (basically the prediction step of LwP where we "greedily" set each y_n to the class prototype that $\boldsymbol{x}_n$ is the closest to). We alternate between these two steps till convergence. This is basically what K-means clustering algorithm does which is essentially the unsupervised counterpart of LwP. :-)