HW3 - ML in HealthCare

1 Part I: Theory

1.1 Clustering

a. The function to minimize is

$$L(\mu) = \sum_{i=1}^{m} |x_i - \mu|$$

The goal is to find the value of μ in the dataset that minimizes this sum.

The derivative of the absolute value function |x| is

$$d/dx \mid x \mid = sign(x)$$

Where

sign(x) = +1 if x>0, -1 if x<0, 0 if x = 0.

For the function, $L(\mu)$, the derivative is

$$d\mu/dL = \sum_{i=1}^{m} sign(xi - \mu).$$

This equation shows how the loss changes as μ moves.

If μ is smaller than the median, more data points satisfy xi > μ, which makes

$$d\mu/dL > 0.$$

This means the loss decreases as μ moves closer to the median.

If μ is larger than the median, more data points satisfy xi < μ, which makes

$$d\mu/dL < 0.$$

This means the loss increases as μ moves away from the median.

At the median, the number of points for which xi>μ equals the number of points for which xi<μ. This makes

$$d\mu/dL = 0.$$

proving that the median minimizes the loss.

The median minimizes the loss because it balances the distances to all the points in the dataset. Any movement of μ away from the median increases the total distance.

The condition for the median to be a minimum is that the derivative $d\mu/dL$ changes sign around it.

For μ smaller than the median, $dL/d\mu > 0$.

For μ larger than the median, $dL/d\mu$ > 0.

This ensures the median is the minimum point of L(μ).

b. To find the value of μ that minimizes L(μ), we begin by calculating its derivative with respect to μ.

Start by considering a single term in the summation $(xi - \mu)^2$. Using the chain rule, the derivative is:

$$d/d\mu(xi - \mu)^2 = 2(xi - \mu) \cdot d/d\mu(-\mu).$$

The derivative of −μ-\mu−μ with respect to μ is −1, so:

$$d/d\mu(xi - \mu)^2 = -2(xi - \mu)$$

Now differentiate the entire summation $L(\mu)$:

$$dL/d\mu = \sum_{i=1}^{m} d/d\mu(xi - \mu)^2$$

Substituting the derivative of each term, we get:

$$dL/d\mu = \sum_{i=1}^{m} -2(xi - \mu)$$

This simplifies:

$$dL/d\mu = -2\sum_{i=1}^{m}(xi - \mu)$$

To simplify further, expand the summation into two separate sums:

$$\sum_{i=1}^{m}(xi - \mu) = \sum_{i=1}^{m} xi - \sum_{i=1}^{m} \mu$$

Since μ is constant across all terms, ∑μ=mμ. Substituting this back, we have:

$$\sum_{i=1}^{m}(xi - \mu) \quad = \sum_{i=1}^{m} xi \ - \ m\mu$$

Replacing this into the derivative, the expression becomes:

$$dL/d\mu = -2\ (\sum_{i=1}^{m} xi \ - \ m\mu)$$

To find the minimum, set $dL/d\mu = 0$.

$$-2\ (\sum_{i=1}^{m} xi \ - \ m\mu) \quad = \ 0$$

Dividing through by −2, we obtain:

$$(\sum_{i=1}^{m} xi \ - \ m\mu) \quad = \ 0$$

We find:

$$\sum_{i=1}^{m} xi \quad = \ m\mu$$

Solving for μ, divide both sides by m:

$$\mu \ = \ \frac{\sum_{i=1}^{m} xi}{m}$$

This shows that the value of μ that minimizes L(μ) is the mean of the data points.

c.

Yes, K-medoid is more robust to noise and outliers than K-means. This is because K-means uses the mean as the cluster center, which is highly sensitive to outliers. Outliers can pull the centroid toward them, distorting the cluster. K-medoid, on the other hand, uses actual data points as the cluster center (medoids). It minimizes the total absolute distance (L1) between the medoid and other points in the cluster. This makes it less affected by extreme values, as outliers are unlikely to be chosen as medoids unless they significantly reduce the total distance. K-medoid is better at handling noise and outliers because it focuses on minimizing distances without being distorted by large, unusual values.

d. K-means is more widely used than K-medoid primarily because it is computationally faster and easier to implement for large datasets. The K-means algorithm calculates the centroid as the mean of points in a cluster, which involves simple mathematical operations that scale well with the size of the data.

K-medoid, on the other hand, involves pairwise distance computations between points to determine the medoid, making it computationally expensive, especially for large datasets. This added complexity makes K-medoid less practical in scenarios where speed and scalability are critical.

Additionally, the K-means algorithm is widely supported in machine learning libraries and tools, making it more accessible to practitioners. Despite its sensitivity to outliers, their efficiency and simplicity often make it a preferred choice when dealing with clean datasets or when speed is a priority.

1.2 SVM:

1. For the ***linear kernel with C = 0.01***, the model allows more mistakes because it focuses on creating a wider margin. This means the decision boundary will be simple and straight but might not separate the points well. Looking at the images, **F** matches this description because it has a straight boundary and several misclassified points.

2. For the ***linear kernel with C = 1***, the model reduces mistakes more aggressively, so the decision boundary still stays straight but does a better job of separating the points. Image **E** fits this because it's linear like **F**, but fewer points are misclassified.

3. Now for the ***2nd order polynomial kernel***, this kernel allows for a curved decision boundary, which helps separate the data better than a straight line. Image **D** shows a smooth, slightly curved boundary, so this must be the match.

4. The ***10th order polynomial kernel*** creates a very complex and wavy decision boundary. It's trying to perfectly separate the points, which often leads to overfitting. Image **C** is the best match here because its boundary looks extremely wavy and complicated.

5. Next, the ***RBF kernel with γ=0.2*** makes smooth boundaries that consider a broader area of the data. It doesn't try to fit the data too tightly. Image **B** fits this description because its boundaries are smooth and general.

6. Finally, the ***RBF kernel with γ=1*** focuses more on individual points, creating tighter and sharper boundaries around them. This matches **A**, where the boundaries look very sharp and tightly follow the data points.

1.3 Kernel functions

a.

The inner product between $\Phi(x1)$ *and* $\Phi(x2)$ is defined as:

$$\langle \Phi(x1),\Phi(x2)\rangle = \sum_{m=0}^{\infty} \phi_m(x_1) \cdot \phi_m(x_2)$$

Substituting the definition of ϕm(x), we get:

$$\phi_m(x_1) \cdot \phi_m(x_2) = \left(\frac{x_1^m}{\sigma^m \sqrt{m!}} e^{-\frac{x_1^2}{2\sigma^2}} \right) \cdot \left(\frac{x_2^m}{\sigma^m \sqrt{m!}} e^{-\frac{x_2^2}{2\sigma^2}} \right).$$

simplify the product:

$$\phi_m(x_1) \cdot \phi_m(x_2) = \frac{(x_1x_2)^m}{\sigma^{2m} m!} e^{-\frac{x_1^2}{2\sigma^2}} e^{-\frac{x_2^2}{2\sigma^2}}$$

Now, sum over all mmm to compute the inner product:

$$\langle \Phi(x1), \Phi(x2)\rangle = e^{-\frac{x_1^2}{2\sigma^2}} e^{-\frac{x_2^2}{2\sigma^2}} \sum_{m=0}^{\infty} \frac{(x_1x_2)^m}{\sigma^{2m} m!}.$$

Notice that the summation $\sum_{m=0}^{\infty} \frac{(x_1x_2)^m}{m!}$, the Taylor series expansion for the exponential function. The exponential function can be written as:

$$\sum_{m=0}^{\infty} \frac{z^m}{m!} = e^z$$

$Here, z = \frac{x_1x_2}{\sigma^2}$, so, the summation becomes:

$$\sum_{m=0}^{\infty} \frac{(x_1x_2)^m}{\sigma^{2m} m!} = e^{\frac{x_1x_2}{\sigma^2}}$$

Substituting this result into the inner product gives:

$$\langle \Phi(x1), \Phi(x2) \rangle = e^{-\frac{x_1^2}{2\sigma^2}} e^{-\frac{x_2^2}{2\sigma^2}} e^{\frac{x_1 x_2}{\sigma^2}}$$

Combine the exponential terms:

$$\langle \Phi(x1), \Phi(x2) \rangle = e^{-\frac{1}{2\sigma^2}(x_1^2 + x_2^2 - 2x_1x_2)}.$$

Next, observe that the expression$(x_1^2 + x_2^2 - 2x_1x_2)$ simplifies to $(x_1 - x_2)^2$. Substituting this back:

$$\langle \Phi(x1), \Phi(x2) \rangle = e^{-\frac{(x_1 - x_2)^2}{2\sigma^2}}.$$

This shows that the inner product can be expressed as a kernel function:

$$K(x1, x2) = e^{-\frac{(x_1 - x_2)^2}{2\sigma^2}}.$$

The kernel depends only on the difference (x1−x2), as required.

Finally, check for symmetry. Observe that:

$$K(x_1, x_2) = e^{-\frac{(x_1 - x_2)^2}{2\sigma^2}} = e^{-\frac{(x_2 - x_1)^2}{2\sigma^2}} = K(x_2, x_1).$$

This confirms that the kernel is symmetric. We have thus shown that the inner product in the transformed space can be written as a kernel function $K(x_1, x_2)$, which depends only on the difference x1−x2 and is symmetric.

b.

This kernel function can be considered a similar function because it measures how close two points, x1x_1x1 and x2x_2x2, are in the transformed space.

The function $K(x_1, x_2) = e^{-\frac{(x_1 - x_2)^2}{2\sigma^2}}$ decreases as the distance (x1−x2) increases. When x1 and x2 are very similar, the function outputs a value close to 1. When they are far apart, it outputs a value closer to 0. This behavior makes it a measure of similarity based on the distance between the points.

c.

The parameter $\gamma = \frac{1}{2\sigma^2}$ controls how the kernel function reacts to the distance between two points. In the formula $K(x1, x2) = e^{-\gamma(x_1 - x_2)^2}$,

γ determines how quickly the similarity decreases as the points move farther apart.

When γ is large, the kernel becomes very sensitive to small changes in the distance between points. It focuses too much on individual points, which can cause overfitting. The model will fit the data too closely and may include noise instead of learning general patterns.

When γ is small, the kernel is less sensitive to the distance. This creates smoother boundaries, but it can cause underfitting. The model becomes too simple and may miss important patterns in the data.

In short, γ controls the balance between overfitting and underfitting, which is important for good model performance.

d.

The Gaussian kernel measures similarity based on the distance between two points. If two points are close, their similarity is high, and if they are far apart, their similarity is low. It focuses on the actual spatial distance in the feature space.

Cosine similarity, on the other hand, measures similarity based on the angle between two points. It ignores the magnitude of the points and only considers their direction. Even if two points are far apart but point in the same direction, their cosine similarity will be high.

In short, the Gaussian kernel depends on distance, while cosine similarity depends on direction.

1.4 Generalization capability

a. The scientific term for the balance Einstein referred to, and which is embedded in BIC, is ***Occam's razor***. It means choosing the simplest model that explains the data without unnecessary complexity.

b. The two terms in BIC, $pln(n)$ and $-2\ln(\hat{L})$, balance simplicity and how well the model fits the data. The first term, $pln(n)$, adds a penalty for model complexity. If the model has too many parameters (p) or the dataset is large (n), this term increases and discourages using overly complex models. The second term, $-2\ln(\hat{L})$, rewards how well the model fits the data. A higher likelihood $(\hat{L})$ means the model explains the data better, which reduces the BIC score and makes the model more favorable.

c. If the balance is violated, two things can happen:

If the model is too simple, it may underfit the data, meaning it won't capture important patterns and will perform poorly.

If the model is too complex, it may overfit the data, meaning it will fit noise or random variations and won't generalize well to new data.

d. With BIC, we aim for a low score. A lower BIC indicates a model that balances simplicity and good fit to the data. A high BIC suggests the model is either too complex or doesn't fit the data well. The goal is to minimize BIC to find the best model.

e. The difference between BIC and AIC is in how they handle model complexity and dataset size.

BIC adds a penalty based on both the number of parameters (p) and the size of the dataset (n) using ln(n). This means the penalty is larger for bigger datasets, so BIC prefers simpler models as the dataset gets bigger.

AIC, on the other hand, only uses 2p as the penalty. It doesn't consider the dataset size, so it's less strict than BIC, especially with large datasets.

In short, BIC is stricter and favors simpler models when there's a lot of data, while AIC is more focused on fitting the data and is less harsh on complexity. Both aim to balance simplicity and accuracy but in slightly different ways.

f. BIC and AIC might choose different models when the dataset is large. Since BIC includes a penalty based on dataset size, it is stricter about model complexity. In a large dataset, BIC could prefer a simpler model, while AIC might select a more complex one because it doesn't consider dataset size. This happens because BIC focuses more on simplicity, and AIC focuses more on fitting the data.

g. Choosing between AIC and BIC depends on a few things.
If the dataset is large, BIC is usually better because it avoids overly complex models.
For smaller datasets, AIC can be better because it is less strict about complexity.
If your goal is to make predictions and fit the data well, AIC is a good choice since it tends to prefer more detailed models.
On the other hand, if you want a simpler model that generalizes better and avoids overfitting, BIC is the way to go. It also depends on whether you care more about model simplicity or accuracy for your specific problem.

1.5 Linear binary classifier with different cost function

a. To find the optimal w, we need to minimize the loss function L(w).

The gradient of L(w) is already given as:

$$\frac{\partial L}{\partial w} = -\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - x_i^T w) + 2\lambda w$$

To minimize the loss, we set the gradient to zero:

$$-\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - x_i^T w) + 2\lambda w = 0$$

Rearranging terms gives:

$$\frac{2}{n}\sum_{i=1}^{n} x_i y_i = \frac{2}{n}\sum_{i=1}^{n} x_i x_i^T w + 2\lambda w$$

Divide through by 2 to make it simpler:

$$\frac{1}{n}\sum_{i=1}^{n} x_i y_i = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T w + \lambda w$$

Instead of summing manually, we use matrix notation to simplify further.

Let X be a matrix where each row is x_i^T, and let Y be the vector of labels y_i. Using this, we can write:

$$\frac{1}{n}X^T Y = \frac{1}{n}X^T X w + \lambda w$$

Multiply both sides by n to remove the fraction:

$$X^T Y = X^T X w + n\lambda w$$

Now group the terms involving w:

$$X^T Y = (X^T X + n\lambda I)w$$

Finally, isolate w by multiplying both sides by the inverse of $(X^T X + n\lambda I)$:

$$w = (X^T X + n\lambda I)^{-1} X^T Y$$

This formula gives the best w by finding a balance between fitting the data and avoiding overfitting. The term $X^T Y$ shows how the features and labels are related, while nλI keeps w small to make the model simpler.

b. Without regularization, the optimal www is:

$$w = (X^T X)^{-1} X^T Y$$

With regularization, the formula changes to:

$$w = (X^T X + n\lambda I)^{-1} X^T Y$$

The difference is the extra term nλI, which prevents www from becoming too large. It also makes the matrix $X^T X + n\lambda I$ easier to invert, even if $X^T X$ is problematic. The regularization term helps control overfitting by keeping w smaller, making the model simpler and better at generalizing new data. If λ is too big, though, the model can underfit and miss important patterns.

c. To update the weights after the first batch, we use this rule:

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial L}{\partial w}$$

We start with the initial weights w0, which are chosen randomly. Then, we calculate the gradient of the loss for the batch of size 8:

$$\frac{\partial L}{\partial w} = -\frac{2}{8} \sum_{i=1}^{8} x_i (y_i - x_i^T w_0) + 2\lambda w_0$$

This formula has two parts: the first part accounts for the error between the predictions and true values for the 8 points in the batch, and the second part is the regularization term λ.

Next, we update the weights using:

$$w_1 = w_0 - \eta \frac{\partial L}{\partial w}$$

Simplifying this, we get:

$$w_1 = w_0 + \frac{\eta}{4} \sum_{i=1}^{8} x_i (y_i - x_i^T w_0) - 2\eta\lambda w_0$$

So, to find the updated weights w1, you apply this formula, using the specific values for the batch, the learning rate η, and the regularization term λ.

d. We are solving the optimal w using the given dataset D with n=3 and λ=1.

The loss function includes a regularization term:

$$L(w) = \frac{1}{3}\sum_{i=1}^{3}(y_i - w^T x_i)^2 + \| w \|^2$$

The formula for the optimal www is:

$$w = (X^T X + n\lambda I)^{-1} X^T Y$$

We calculate the matrices first. The feature matrix X and label vector Y are:

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 2 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

The formula for w is:

$$w = (X^T X + n\lambda I)^{-1} X^T Y.$$

We calculate each term step by step.

Compute $X^T X$:

$$X^T X = \begin{bmatrix} 6 & 1 \\ 1 & 1 \end{bmatrix}$$

Compute $X^T Y$:

$$X^T Y = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Compute $n\lambda I (n = 3, \lambda = 1)$:

$$n\lambda I = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$$

Add $X^T X$ + nλI =

$$\begin{bmatrix} 6 & 1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} 9 & 1 \\ 1 & 4 \end{bmatrix}$$

Finally, calculate w:

$$w = \frac{1}{35}\begin{bmatrix} 4 & -1 \\ -1 & 9 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{35}\begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

$$w = \begin{bmatrix} \frac{3}{35} \\ \frac{8}{35} \end{bmatrix}$$

The decision boundary is defined $as\ w^T x = 0$, which simplifies to:

$$\frac{3}{35}x_1 + \frac{8}{35}x_2 = 0.$$

Solving for x2:

$$x_2 = -\frac{3}{8}x_1$$

To check if the decision boundary separates the points correctly:

For x = [1,0], y = +1, $w^T x = 3/35 > 0$.

For x = [1,1], y = +1, $w^T x = 3/35 + 8/35 = 11/35 > 0$.

For x = [2,0], y = -1, $w^T x = 6/35 > 0$.

The decision boundary does not perfectly separate the data points. One negative example is misclassified. This shows that using squared loss minimizes the overall error but might not be ideal for classification tasks.