<u>HW3 – Dry Part – Nir Sassy 316491737</u>

<u>1.1 – Clustering</u>

a. Assuming m (the number of examples) is an odd number, and $x \in R^1$ (a scalar), we want to find μ (the cluster medoid) which will minimize the sum of the absolute distances of all the samples from the medoid ($L_1\ norm$), given μ belongs to the dataset. In other words, we have the following optimization problem:

$$\hat{\mu} = \underset{\mu}{\operatorname{argmin}} \sum_{i=1}^{m} |x_i - \mu|$$

Taking the derivative with respect to μ yields (As mentioned in the question, we'll use $\frac{d}{dx}|x| = sign(x)$ with no further explanation):

$$\frac{d}{d\mu} = -1 * \sum_{i=1}^{m} sign(x_i - \mu)$$

After comparing to zero to find the minimum, we get:

$$\sum_{i=1}^{m} 1_{x_i > \mu}, -1_{x_i < \mu} = 0$$

That means that for every sample x_i that is bigger than μ we'll add 1, and for every sample x_i that is smaller than μ we will add -1. Therefore, this term will equal zero only for μ which satisfies that half of the samples are above it and half are below it, hence the **median**.

A sufficient condition for the median to be a minimum is that there are an odd number of examples. That's because an even number of examples would result in a median value than is not part of the dataset (In this case the median would be the mean of the 2 examples in the middle of the sorted data) and therefore cannot be chosen as a medoid in K-medoids algorithm.

b. For K-means, we have the following optimization problem:

$$\hat{\mu} = \underset{\mu}{\operatorname{argmin}} \sum_{i=1}^{m} (x_i - \mu)^2$$

Whereas $x \in R^1$ (a scalar), and μ is the cluster centroid. Looking at the loss function $L(\mu)$:

$$L(\mu) = \sum_{i=1}^{m}(x_i - \mu)^2$$

Taking the derivative with respect to μ and comparing to zero yields:

$$L'(\mu) = -2 * \sum_{i=1}^{m}(x_i - \mu) = 0$$

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

μ is not connected to the sum index i and therefore we can extract it from the sum term:

$$-m \cdot \mu + \sum_{i=1}^{m} x_i = 0$$

$$\hat{\mu} = \frac{1}{m} \cdot \sum_{i=1}^{m} x_i$$

We got that the centroid μ that'll minimize our loss function is the mean of the samples.

Lastly, we'll check that the $\hat{\mu}$ we got is indeed a minimum:

$$L''(\mu) = \left(-2(-m \cdot \mu) + \sum_{i=1}^{m}(x_i)\right)' = 2m > 0 => \boldsymbol{minimum}$$

c. In K-Medoids, the medoids update process is based on the median of the samples, while the K-Means updates the centroids based on the mean of the samples. The median is less sensitive to noise or outliers than the mean, making the K-medoid algorithm less sensitive to noise or outliers.

<u>1.2</u> <u>SVM</u>

- First, we can clearly see that figures A and D presents linear classifiers, because the decision boundary is a linear line. C parameter is a regularization parameter which decides how much do we penalize for misclassification errors, and thus affects the trade-off between maximizing the margin between the decision boundary and the support vectors (the optimization problem without the regularization term) and the amount of misclassification errors. Therefore, I'd say that figure A is a linear kernel SVM with C = 1, as we can see that although there are no errors, the margin size is small hinting there was a high penalty for errors during training and the algorithm eventually converged to this solution. Figure D is a linear kernel SVM with C = 0.01, as there are no misclassification errors at all but also there is a big margin compared to figure A, hinting that the algorithm wasn't penalized much during training and was able to find the higher margin as a solution.

- Second, looking at figure B and E, we can see that the decision boundary for the blue labeled samples have a circular-like shape. This shapes clearly represents an RBF (Radial Basis Function) kernel. As said in the lectures, the gamma coefficient decides how much we fit our data, meaning that high gamma will lead to overfitting. In this case, we can easily see that figure B's decision boundary is almost perfectly fitted on the blue labeled samples, hinting there was a high gamma coefficient during training, while figure E's decision boundary is a more generalized and less fitted decision boundary, hinting there was a low coefficient during training. Therefore, I'd say that Figure B represents an RBF kernel with $\gamma = 1$, and figure E represents an RBF kernel with $\gamma = 0.2$.
- Lastly, Figure C and F represent a polynomial kernel, because (that's what we left with) the decision boundary are a curved lines fitting for a polynomial term. In this case, we can clearly see that figure F's decision boundary is highly overfitted and has a complex shape, hinting on a kernel with high polynomial order, while figure C's decision boundary is a simple and generalized boundary hinting on a kernel with low polynomial order. Therefore, I'd say that Figure C is for a polynomial kernel of order 2, and Figure F is for a polynomial kernel of order 10.

In summary:

Figure A – linear kernel SVM with C = 1, Figure B – RBF kernel SVM with $\gamma = 1$, Figure C – 2^{nd} Polynomial kernel SVM, Figure D – linear kernel SVM with C = 0.01, Figure E – RBF kernel SVM with $\gamma = 0.2$, Figure F – 10^{th} Polynomial kernel SVM.

1.3 – Kernel Function

The inner product $\langle\Phi(\mathrm{x}_1),\Phi(x_2)\rangle = \Phi(x_1)^T\Phi(x_2)$ can be written explicitly as follows:

$$\Phi(x_1)^T\Phi(x_2) = \begin{pmatrix}\phi_0(x_1)\\ \vdots \\ \phi_m(x_1)\end{pmatrix}\begin{pmatrix}\phi_0(x_2) & \cdots & \phi_m(x_2)\end{pmatrix} =$$

$$= \phi_0(x_1)\phi_0(x_2) + \cdots + \phi_m(x_1)\phi_m(x_2) =$$

$$= \frac{x_1^0}{\sqrt{0!}}e^{-\frac{x_1^2}{2}} \cdot \frac{x_2^0}{\sqrt{0!}}e^{-\frac{x_2^2}{2}} + \cdots + \frac{x_1^m}{\sqrt{m!}}e^{-\frac{x_1^2}{2}} \cdot \frac{x_2^m}{\sqrt{m!}}e^{-\frac{x_2^2}{2}} =$$

$$= e^{-\frac{x_1^2+x_2^2}{2}}\left(1 + x_1x_2 + \frac{(x_1x_2)^2}{2!} + \cdots + \frac{(x_1x_2)^m}{m!}\right) =$$

Notice that the term inside the brackets is the Taylor series of $e^{x_1x_2}$:

$$= e^{-\frac{x_1^2+x_2^2}{2}} \cdot e^{x_1x_2} = e^{\frac{-x_1^2-x_2^2+2x_1x_2}{2}} =$$

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

Also, we notice that $K(x_1,x_2) = K(x_2,x_1)$:

$$K(x_1,x_2) = e^{\frac{-(x_1-x_2)^2}{2}} = e^{\frac{-(-1(x_2-x_1))^2}{2}} = e^{\frac{-(-1)^2(x_2-x_1)^2}{2}} = e^{\frac{-(x_2-x_1)^2}{2}} =$$

$$= K(x_2,x_1).$$

1.4 Generalization Capability

a. From ML point of view, the scientific term of balance that Einstein meant to is the **Bias-Variance Tradeoff**, in which a model with good **balance** between bias and variance will have the capability to generalize for unseen data that is equally distributed to data that it has seen and trained on.

b. 2p refers to the number of parameters in our model, hence the complexity of our model. $2\ln(\hat{L})$ refers to the estimated likelihood given these parameters, hence how fitted our model is to the data. For both terms, higher values mean a more complex model or a more fitted model. In (a) we mentioned that generalization is achieved by finding the balance between bias and variance – A very complex model (a lot of parameters - high

‘2p’ value) with poor fitting (low ’$2\ln(\hat{L})$’ value) would have high bias, and a very simple model with high fitting (which is also the opposite case for the terms values) would have high variance.

c. The two options are overfitting and underfitting. A model with high bias is considered under-fitted, meaning it will not generalize good because it has not learned the “general trend” of the data distribution from the training set, let alone data it has never seen. A model with high variance is considered over-fitted, and it will not generalize good as well, because the model is so fitted on the data it has seen that even a minor deviation (that would come up in the data it hasn’t seen) would result in a major error.

d. As said in (b), high value for ‘2p’ and low value for ’$2\ln(\hat{L})$’ would result in high bias, and the opposite case would result in high variance. Therefore, we see that if ‘2p’ and ‘$2\ln(\hat{L})$’ are about the same size, we would find a good balance between bias and variance and have a model that would generalize good. Meaning, the AIC should have an absolute value that is as low as possible, with the optimal case of zero.

1.5 <u>Linear binary classifier with different cost function</u>

a. To find the optimal weights we’ll need to solve the following optimization problem:

$$\widehat{W} = \underset{W}{\operatorname{argmin}}\, L(W) = \underset{W}{\operatorname{argmin}}\, \frac{1}{n}\sum_{i=1}^{n}(y_i - W^T x_i)^2$$

Taking the derivative with respect to w (this term is given) and comparing to zero:

$$\frac{\partial L}{\partial W} = -\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - x_i^T W) = 0$$

$$\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i x_i^T W = 0$$

We can write this equation in a matrix form, where $X \in R^{dxn}$, $Y \in R^{nx1}$ and $W \in R^{dx1}$:

$$XY - XX^T W = 0$$

Multiplying (from left) by $(XX^T)^{-1}$:

$$W = (XX^T)^{-1}XY$$

We see that the solution is the same as the pseudo-inverse solution we got in the lecture for linear regression, only here it's $(XX^T)^{-1}XY$ instead of $(X^TX)^{-1}X^TY$. The difference is that x_i is a column vector, thus making the X matrix (which is, in this case, a column-concatenation of $x_i's$) in R^{dxn} instead of R^{nxd} as seen in the course for $x_i's$ that are row vectors. Obviously, these are just different ways to write down our data, and the resulted weights are the same for both cases.

b. The weights updating process, as defined in the gradient descent algorithm, is as follows:

$$W_{n+1} = W_n - \eta \cdot \frac{\partial L}{\partial W}$$

If so, the updated set of weights after the first batch with size of 4 would be:

$$W_1 = W_0 - \eta \cdot \frac{2}{4} \cdot (-1) \sum_{i=1}^{4} x_i(y_i - x_i^T W_0) =$$

$$= W_0 + \eta \cdot \frac{1}{2} \sum_{i=1}^{4} x_i(y_i - x_i^T W_0)$$

(W_0 is the initial weights vector, and W_1 is the weights vector after one iteration.)

c. We can use the solution found in (a) to jump straight to the optimal weights, without using an iterative algorithm such as gradient descent (gradient descent is used when we cannot find a close-form solution for the loss function minimum).

First, we will write the given dataset as matrices:

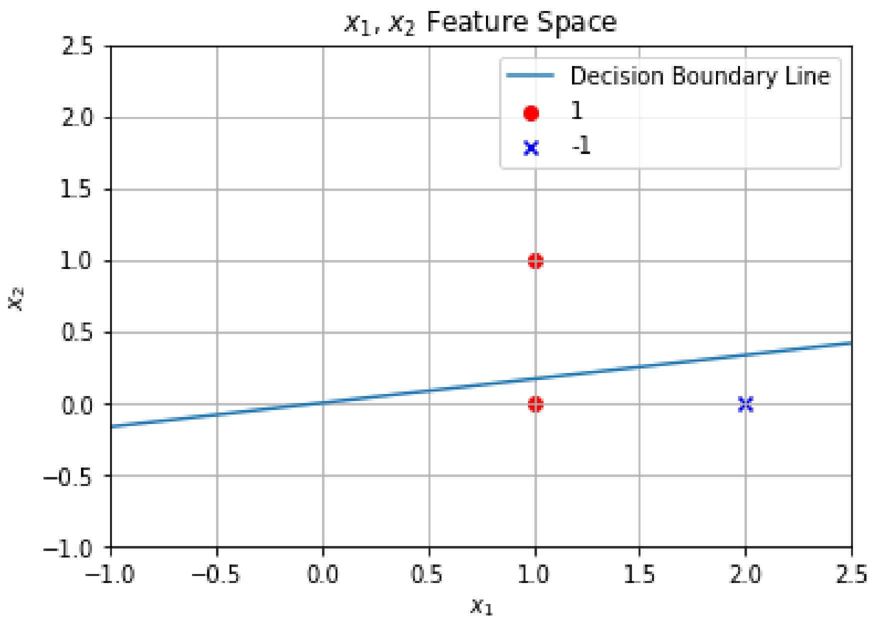
$$X = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}$$

Therefore, the optimal weights are given by:

$$W = \left(\begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \end{pmatrix}^T \right)^{-1} \begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} \underbrace{=}_{python} \begin{pmatrix} -0.2 \\ 1.2 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$$

Where w_1 is the weight of feature x_1, and w_2 is the weight of feature x_2. The boundary decision line is given by $W^T x = 0, or\ w_1 x_1 + w_2 x_2 = 0$. For the optimal weights found, we get $-0.2 \cdot x_1 + 1.2 \cdot x_2 = 0 \rightarrow x_2 = \frac{1}{6} \cdot x_1$.

The results are shown in the next figure:



The feature space with the labeled samples and the decision boundary line

The predicted value is defined as $\hat{y} = sign(w^T x)$, which means that every point below the decision boundary line is predicted to be -1, and every point above the decision boundary line is predicted to be 1.

We can see that even for this simple binary classification task of 3 samples, the optimal weights for the decision boundary line given by our MSE cost function have 1 error and resulted in 66.6% accuracy. The reason is that the MSE does not suit for classification problems.

The MSE does not consider that a sample was misclassified, but rather just calculates the error based on the difference between the true label and the predicted label. The cross-entropy cost function, for example, does take it into account by calculating the loss based on the probability of the predicted values, in a way that predicted values that are close (in terms of probability) to the true values would have a smaller loss and so on.

In conclusion, this question emphasizes the importance of choosing a cost function that fits the task, otherwise our results would be garbage.