COMS W4771: Machine Learning (sec:001) - Homework #1

Name: Geraldi Dzakwan (gd2551). Discussants: Deka (da2897), Patricia (pk2618)

September 23, 2019

Problem 1

I aspire to be a data scientist or machine learning researcher once I graduate from Columbia as my career value. Therefore, I want to learn machine learning by specifically taking this course, in hope that the followings can be fulfilled:

1. I will have a strong mathematical foundation for machine learning so that I can be more than a user. I don't want to just use machine learning libraries without knowing the math (or at least the intuition) behind the techniques that I use.

2. I will have a broad knowledge of machine learning fundamentals to enable me to learn the advanced ones.

3. By having strong mathematical foundation and broad knowledge of machine learning, I aim to be able to read machine learning papers and understand the mathematical idea behind them (not just blindly implementing them in codes). Moreover, I also hope that in times where available models/techniques don't work well on a problem that I face (at work or in a research project), I can design a customized solution for that very specific problem (e.g. tweak a machine learning algorithm, create a customized optimization algorithm, design my own neural network architecture, etc.).

4. I don't know if this class will expose me to wide range of machine learning tools/frameworks through projects or coding assignments. But, if it does, it will definitely help me to familiarize myself with all the technical/programming stuff related to machine learning so that I can easily explore about machine learning on my own outside the class (e.g. working on side projects, participating on Kaggle competition, doing an independent research, etc.).

Problem 2

(a) There are two big steps:

1. Calculate Distance Matrix
   Let's assume we have m test feature vectors and n training feature vectors. We can stack the test vectors as a matrix $T \in \mathbb{R}^{mxd}$ and n training vectors as another $X \in \mathbb{R}^{nxd}$. Using those two matrices, We can derive a matrix $D \in \mathbb{R}^{mxn}$ in which an element d_{ij} in matrix D denotes the Euclidean distance between vector t_i and vector x_j. This is exactly as described in the Problem 1 description and we can write all this as:

$$d_{ij} = \left\| t_i - x_j \right\|^2 = (t_i - x_j)^T (t_i - x_j) = t_i^T t_i + x_j^T x_j - 2 t_i^T x_j$$

   However, in my implementation, instead of using loops and calculating each d_{ij} separately, I operate directly on the matrix instead to speed up the computation. In my implementation, this is how it is done:

   1. To get $t_i^T t_i$ for all i, we can square all the elements in T (test_matrix) and then for each row, we take the sum of the elements. In numpy, we do this by specifying axis=1, that is to sum over rows.
      ```
      s_test = np.sum(np.square(test_matrix), axis=1)
      ```
      This will result in a vector of size m, or in numpy shape it would be one dimensional array with shape: $(m,)$.

   2. To get $x_j^t x_j$ for all j, we do the same as above.
      ```
      s_train = np.sum(np.square(train_matrix), axis=1)
      ```
      This will result in a vector of size n, or in numpy shape it would be one dimensional array with shape: $(n,)$.

   3. To get $t_i^T x_j$ for all i and j, we simply perform matrix multiplication (np.dot) of X and transpose of T:
      ```
      matrix_dot_product = np.dot(test_matrix, train_matrix.T)
      ```
      This will result in a matrix of size m x n, or in numpy shape it would be two dimensional array with shape: (m, n).

   4. Things get tricky when we try to sum all of the three elements above. Basically, from one dimensional array of s_test and s_train, we need to change both of their shapes into a two dimensional array of size (m, n) and then sum them. Think of it as creating a matrix where there is n

columns of m (this is for s_test) and another matrix where there is m rows of n (this is for s_train). In numpy, this can be done conveniently by broadcasting. To do broadcasting, we just tell numpy to extend s_test from 1d array of size m to 2d array of size (m,1). This is done by slicing s_test into s_test[:, None]. When we later try to sum this 2d array of s_test with 1d array of s_train, numpy will automatically "broadcast" both of our arrays to 2d array of size (m, n).

```
s_test_train = s_test[:, None] + s_train
```

Reference: https://numpy.org/devdocs/user/theory.broadcasting.html.

5. Finally, we can sum all of the elements that we have computed above and take the square root of each element in the matrix to form our distance matrix:

```
distance_matrix = np.sqrt(s_test_train - 2*matrix_dot_product)
```

2. Predicting Label
Iterate every row in distance matrix and append the predicted label to an array of prediction. To get the label for each iteration, we do the followings:
   1. Find k-smallest values in the row and get their indexes using np.argpartition.

```
k_smallest_indexes = np.argpartition(dist_vector, k)[:k]
```

   2. Build a dictionary with label as dictionary key and number of its occurrences in k-smallest values as dictionary value.

```
for idx in k_smallest_indexes:
    label = label_vector[idx][0]
    if label not in count_dict:
        count_dict[label] = 1
    else:
        count_dict[label] = count_dict[label] + 1
```

   3. Find label with the most occurrences. If k is even and there are more than one labels with highest count, the "lowest" label is taken, e.g. label '1' is more favorable than lable '2', etc.

```
final_label = max(count_dict, key=lambda key: count_dict[key])
```

(b) For $n = [1000, 2000, 4000, 8000]$, this is what I get:

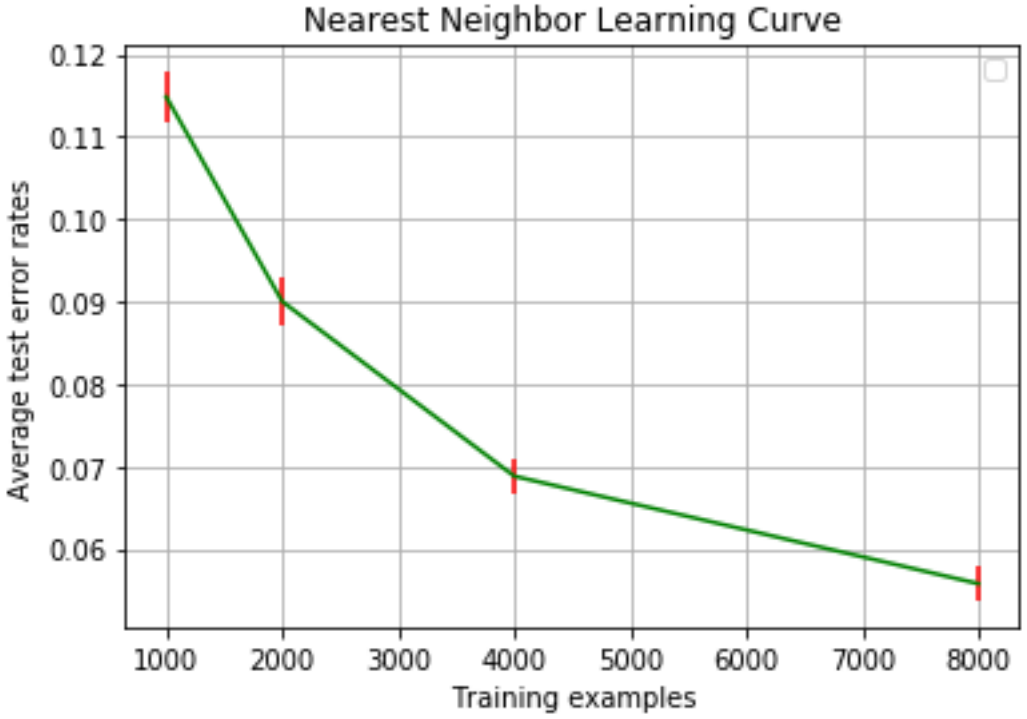

Figure 1: Nearest Neighbor Learning Curve

train_size	*mean_error*	*std*
1000	11.4850%	$0.2984 \text{x} 10^{-2}$
2000	9.0010%	$0.3022 \text{x} 10^{-2}$
4000	6.8820%	$0.2098 \text{x} 10^{-2}$
8000	5.5780%	$0.2041 \text{x} 10^{-2}$

Table 1: Nearest Neighbor Mean Error and Standard Deviation

(c) To do the 10-fold cross validation, I use Python ***sklearn*** library. All training data are used, meaning that for each fold, 54000 data are for training and the remaining 6000 are for validation. Table 2 in the next page shows the cross validation error rate for each k.

From Table 2, we can see the optimal hyperparameter is $\boldsymbol{k = 3}$. Using that value, I use full training data (60000 instances) and get test error rate (tested against 10000 data) of **2.88%**.

k	cv_error_rate
1	2.9500%
2	2.9500%
3	**2.8233%**
4	2.8700%
5	2.9283%
6	2.9533%
7	3.0617%
8	3.1200%
9	3.2300%
10	3.2633%

Table 2: kNN Cross Validation Error Rates

Problem 3

(a) Consider the worst case when all data points that we pick are not amongst the top-5% nearest neighbor data points. Assuming that n is divisible by 20, that would be a sample of size $n - 0.05n = 0.95n$. Thus, in this worst case, in order to get at least one sample from top-5% nearest neighbor data points, we need to increase our sample size by 1. It would give us the following inequation.

$$T \geq (n - 0.05n) + 1$$

$$T \geq 0.95n + 1$$

Now we want to consider the case that n is not divisible by 20. The equation would still be the same, except that we need to take the floor of $0.05n$. I assume that top-5% of, for example, 102, is 5 (the floor) instead of 6 (the ceiling). That would give us the following inequation.

$$T \geq (n - \lfloor 0.05n \rfloor) + 1$$

Let's define $n = 20k + r$ in which k and r are integers within these ranges: $k \geq 0$ and $1 \leq r \leq 19$. This gets us a series of $n > 0$ where n is not divisible by 20. Substitute n and we get:

$$T \geq (20k + r - \lfloor 0.05(20k + r) \rfloor) + 1$$

$$T \geq (20k + r - \lfloor k + 0.05r) \rfloor) + 1$$

As k is an integer, we can get it out from the floor function:

$$T \geq (20k + r - k - \lfloor 0.05r \rfloor) + 1$$

$$T \geq (19k + r - \lfloor 0.05r \rfloor) + 1$$

Since $1 \leq r \leq 19$, then $0.05 \leq 0.05r \leq 0.95$. This means that $\lfloor 0.05r \rfloor$ is zero for any value of r. That gives us:

$$T \geq (19k + r) + 1$$

$$T \geq (19k + 0.95r + 0.05r) + 1$$

$$T \geq (0.95(20k + r) + 0.05r) + 1$$

$$T \geq (0.95n + 0.05r) + 1$$

Notices that $19k + r$ is an integer and so is $0.95n + 0.05r$. Since $0.05r < 1$, that integer would be nothing else but $\lceil 0.95n \rceil$, the smallest integer that is greater than $0.95n$. Then, we get:

$$T \geq \lceil 0.95n \rceil + 1$$

Thus, we finally have:

1. $T \geq 0.95n + 1$ for n divisible by 20
2. $T \geq \lceil 0.95n \rceil + 1$ for n not divisible by 20

For n divisible by 20, $0.95n$ is an integer and hence $0.95n = \lceil 0.95n \rceil$. Thus, we can come up with one inequation instead for all values of n:

$$\boldsymbol{T \geq \lceil 0.95n \rceil + 1}$$

(b) Redefining the problem, we need to prove that the chance of all 90 data points that we pick as a sample are not in the top-5% nearest neighbor data points is at most 1%. In other words, the chance is less than or equal to 1% that those 90 data points are in the rest 95% of the data. Because this is a sampling without replacement, those probability can be modeled as how many combinations are there to get 90 data points from a specific 95% sample of dataset of size n (those that don't contain the top-5%) per how many combinations are there to get 90 data points from a dataset of size n. Translating it into mathematical notation:

$$p = \frac{\binom{\lceil 0.95n \rceil}{90}}{\binom{n}{90}} = \frac{\frac{\lceil 0.95n \rceil!}{(\lceil 0.95n \rceil - 90)!\, 90!}}{\frac{n!}{(n-90)!\, 90!}} = \frac{\lceil 0.95n \rceil\ (n-90)!}{n!\ (\lceil 0.95n \rceil - 90)!} = (\frac{(n-90)!}{n!})(\frac{\lceil 0.95n \rceil!}{(\lceil 0.95n \rceil - 90)!})$$

$$p = \frac{\lceil 0.95n \rceil(\lceil 0.95n \rceil - 1)(\lceil 0.95n \rceil - 2)...(\lceil 0.95n \rceil - 89)}{n(n-1)(n-2)...(n-89)} = \prod_{i=0}^{89} \frac{\lceil 0.95n \rceil - i}{n - i}$$

We want to prove that for $T = 90$, p is less than 0.01 regardless of n (size of the dataset). As the the dataset grows (n goes bigger), p will increase. Thus, we need to consider n as biggest as possible (n goes to ∞) to prove $T = 90$ is sufficient. We can state this as:

$$p_{n \to \infty} = \lim_{n \to \infty} \prod_{i=0}^{89} \frac{\lceil 0.95n \rceil - i}{n - i}$$

Using multiplication rule for limit:

$$p_{n\to\infty} = \prod_{i=0}^{89} \lim_{n\to\infty} \frac{\lceil 0.95n \rceil - i}{n - i}$$

Divide into two cases:

Case 1: $0.95n$ is an integer

$$p_{n\to\infty} = \prod_{i=0}^{89} \lim_{n\to\infty} \frac{\lceil 0.95n \rceil - i}{n - i} = \prod_{i=0}^{89} \lim_{n\to\infty} \frac{0.95n - i}{n - i} = \prod_{i=0}^{89} \lim_{n\to\infty} \frac{0.95 - i/n}{1 - i/n}$$

$$p_{n\to\infty} = \prod_{i=0}^{89} \frac{0.95 - 0}{1 - 0} = \prod_{i=0}^{89} 0.95 = 0.95^{90} = 0.0098883647$$

Case 2: $0.95n$ is not an integer. Use the property that $\lceil x \rceil = \lfloor x \rfloor + 1$.

$$p_{n\to\infty} = \prod_{i=0}^{89} \lim_{n\to\infty} \frac{\lceil 0.95n \rceil - i}{n - i} = \prod_{i=0}^{89} \lim_{n\to\infty} \frac{\lfloor 0.95n \rfloor + 1 - i}{n - i}$$

Use the property that $\lfloor x \rfloor = x - frac(x)$ where $0 < frac(x) < 1$.

$$p_{n\to\infty} = \prod_{i=0}^{89} \lim_{n\to\infty} \frac{0.95n - frac(0.95n) + 1 - i}{n - i}$$

$$p_{n\to\infty} = \prod_{i=0}^{89} \lim_{n\to\infty} \frac{0.95 - frac(0.95n)/n + 1/n - i/n}{1 - i/n}$$

$$p_{n\to\infty} = \prod_{i=0}^{89} \frac{0.95 - 0 + 0 - 0}{1 - 0} = \prod_{i=0}^{89} 0.95 = 0.95^{90} = 0.0098883647$$

Both of the cases yield the same result. Thus, it is proven that:

$$\boldsymbol{p_{n\to\infty} < 0.01}$$

Problem 4

(a) We will consider these two ranges of x:

Case 1: For $x \leq 0$, $f_\theta(x) = 0$. Thus, in this case, the likelihood is always zero and the MLE would be zero as well.

Case 2: For $x > 0$, $f_\theta(x) \propto x^2 e^{-x/\theta}$. This means that $f_\theta(x) = cx^2 e^{-x/\theta}$ for some constant c: $0 < c < \infty$. Using the property that probability density functions should integrate to one, we can determine c:

$$c \int_0^\infty x^2 e^{-x/\theta} = 1$$

Using integration by parts, taking $u = x^2$ and $dv/dx = -\theta e^{-x/\theta}$, we get:

$$-\theta e^{-x/\theta} x^2 \Big|_0^\infty - \int_0^\infty (2x)(-\theta e^{-x/\theta}) = 1/c$$

$$-\theta e^{-x/\theta} x^2 \Big|_0^\infty + 2\theta \int_0^\infty x e^{-x/\theta} = 1/c$$

Another integration by parts, taking $u = x$ and $dv/dx = -\theta e^{-x/\theta}$, we get:

$$-\theta e^{-x/\theta} x^2 \Big|_0^\infty + 2\theta(-\theta e^{-x/\theta} x \Big|_0^\infty + \int_0^\infty \theta e^{-x\theta}) = 1/c$$

$$-\theta e^{-x/\theta} x^2 \Big|_0^\infty + 2\theta(-\theta e^{-x/\theta} x \Big|_0^\infty - \theta^2 e^{-x/\theta} \Big|_0^\infty) = 1/c$$

$$-\theta e^{-x/\theta} x^2 \Big|_0^\infty - 2\theta^2 e^{-x/\theta} x \Big|_0^\infty - 2\theta^3 e^{-x/\theta} \Big|_0^\infty = 1/c$$

$$-\theta e^{-x/\theta} x^2 - 2\theta^2 e^{-x/\theta} x - 2\theta^3 e^{-x/\theta} \Big|_0^\infty = 1/c$$

1. Calculate upper bound:

$$\text{upper\_bound} = -\theta \lim_{x\to\infty} \frac{x^2}{e^{x/\theta}} - 2\theta^2 \lim_{x\to\infty} \frac{x}{e^{x/\theta}} - 2\theta^3 \lim_{x\to\infty} \frac{1}{e^{x/\theta}}$$

Using L'Hospital theorem a few times, we get:

$$\text{upper\_bound} = -2\theta^2 \lim_{x\to\infty} \frac{x}{e^{x/\theta}} - 2\theta^3 \lim_{x\to\infty} \frac{1}{e^{x/\theta}} - 2\theta^3(0)$$

$$\text{upper\_bound} = -2\theta^3 \lim_{x\to\infty} \frac{1}{e^{x/\theta}} - 2\theta^3(0) - 2\theta^3(0)$$

$$\text{upper\_bound} = -2\theta^3(0) - 2\theta^3(0) - 2\theta^3(0) = 0$$

2. Calculate lower bound:

$$\text{lower\_bound} = -\theta \lim_{x\to 0+} \frac{x^2}{e^{x/\theta}} - 2\theta^2 \lim_{x\to 0+} \frac{x}{e^{x/\theta}} - 2\theta^3 \lim_{x\to 0+} \frac{1}{e^{x/\theta}}$$

$$\text{lower\_bound} = -\theta(\frac{0}{1}) - 2\theta^2(\frac{0}{1}) - 2\theta^3(\frac{1}{1}) = -2\theta^3$$

$$\text{Finally, upper\_bound} - \text{lower\_bound} = 1/c$$

$$0 - (-2\theta^3) = 1/c$$

$$c = 1/(2\theta^3)$$

Hence, the likelihood for $\{x_1, x_2, x_3, ..., x_n\}$ would be:

$$L(\theta|x) = \prod_{i=1}^{n} f_\theta(x_i) = \prod_{i=1}^{n} (c x_i^2 e^{-x_i/\theta}) = \prod_{i=1}^{n} (\frac{x_i^2 e^{-x_i/\theta}}{2\theta^3})$$

Take the log likelihood instead so it's easier to expand:

$$\ln L(\theta|x) = \ln(\prod_{i=1}^{n} f_\theta(x_i)) = \sum_{i=1}^{n} \ln(f_\theta(x_i)) = \sum_{i=1}^{n} \ln(\frac{x_i^2 e^{-x_i/\theta}}{2\theta^3})$$

$$\ln L(\theta|x) = \sum_{i=1}^{n} \ln x_i^2 + \sum_{i=1}^{n} \ln e^{-x_i/\theta} - \sum_{i=1}^{n} \ln 2\theta^3$$

$$\ln L(\theta|x) = 2\sum_{i=1}^{n} \ln x_i + \sum_{i=1}^{n} (-x_i/\theta) \ln e - \ln 2\theta^3$$

$$\ln L(\theta|x) = 2\sum_{i=1}^{n} \ln x_i - \frac{1}{\theta}\sum_{i=1}^{n} x_i - \ln 2\theta^3$$

To minimize this, take its first derivative and find the θ that makes it zero:

$$\frac{d}{d\theta} \ln L(\theta|x) = 0$$

$$0 + \frac{1}{\theta^2}\sum_{i=1}^{n} x_i - \frac{6\theta^2}{2\theta^3} = 0$$

$$\frac{3}{\theta} = \frac{1}{\theta^2}\sum_{i=1}^{n} x_i \to \theta = \frac{1}{3}\sum_{i=1}^{n} x_i$$

Since $x > 0$, this MLE formula will always be greater than zero, thus greater than MLE for case 1. Hence, we can pick this as our MLE formula.

$$\text{Thus, the MLE formula is } \boldsymbol{\hat{\theta}_{mle} = \frac{1}{3}\sum_{i=1}^{n} x_i}$$

(b) We will consider these two ranges of x:

Case 1: For $x < 0$ and $x > \theta$, $f_\theta(x) = 0$. Thus, in this case, the likelihood is always zero and the MLE would be zero as well.

Case 2: For $0 \leq x \leq \theta$, $f_\theta(x) \propto 1$. This means that $f_\theta(x) = c$ for some constant c: $0 < c < \infty$. Using the property that probability density functions should integrate to one, we can determine c:

$$c \int_0^\theta dx = 1$$

$$cx\Big|_0^\theta = 1$$

$$c[\theta - 0] = 1$$

$$c = 1/\theta$$

Hence, the likelihood for $\{x_1, x_2, x_3, ..., x_n\}$ would be:

$$L(\theta|x) = \prod_{i=1}^{n} f_\theta(x_i) = c^n = (1/\theta)^n = 1/\theta^n$$

To maximize that likelihood, we want to choose the smallest θ possible. Since we know that $0 \leq x \leq \theta$, the smallest θ possible would be the the highest x in the set. Thus, the MLE formula would be: $\max(\{x_1, x_2, x_3, ..., x_n\})$. Since $0 \leq x \leq \theta$ and $\theta > 0$, this MLE formula will always be greater or equal to MLE for case 1 (≥ 0). Hence, we can pick this as our MLE formula.

Thus, the MLE formula is $\boldsymbol{\hat{\theta}_{mle} = \max(\{x_1, x_2, x_3, ..., x_n\})}$

Problem 5

(a) We can create a line of $x_1 = 2$ so that all the blue dots (positive examples) in the training data are on the right side of the line, hence training false negative rate is zero. By drawing this line, there will be 3 false positives in the training data. Those are 3 red dots (negative examples) on the right side of the line. Thus, exactly the training error is exactly $3/18 = 1/6$. In this case, θ for x_1 will be 2, meanwhile for x_2 we don't need any θ. This tree has depth equals to 1 (smallest) and can be coded as below.

```
def d_tree_a(x1, x2):
    if x1 < 2:
        return 0
    else:
        return 1
```

(b) To make training rate equals to zero, we need to at least make a decision tree of depth 2. This is because there is no line $x_1 = \theta_1$ or $x_2 = \theta_2$ that can fully separate the red and blue dots. We can start with the previous value of $\theta_1 = 2$. Here, to make training rate equals to zero, we can pick any θ_2 value between some value below 1 (probably around 0.6 by looking at the the picture) and some value below 2.2 (also by looking at the picture).

To fulfill the second condition (test error rate as high as possible), we need to look at the distribution of the test data. We can see that the test data distribution is exactly the same as training data except that there are many blue dots in the range of $3 < x_1 < 4$ and $1 < x_2 < 2$ in which they don't occur in the training data. If we want to maximize the test error, we should pick θ_2 outside the range of $1 < x_2 < 2$ so all the blue dots in the range of $3 < x_1 < 4$ and $1 < x_2 < 2$ will be classified as false negative. Thus, we can just pick $\theta_2 = 2$. This tree has depth equals to 2 (smallest) and can be coded as below.

```
def d_tree_b(x1, x2):
    if x1 < 2:
        return 0
    else:
        if x2 < 2:
            return 0
        else:
            return 1
```

Problem 6

(a) Let X is random variable that denotes number of heads occurring in two coin tosses ($X = [0, 1, 2]$). Then, we can define two probabilities below:

$$P(X = 0) = P(tails).P(tails) = (1/2)(1/2) = 1/4$$

$$P(X \geq 1) = 1 - P(X = 0) = 1 - (1/2)(1/2) = 3/4$$

Using those probabilities, $Y_1 = 1$ occurs when a person that has property P (θ) gets at least one head $P(X \geq 1)$ or when a person that doesn't have property P ($1 - \theta$) get no heads $P(X = 0)$. Thus, $P(Y_1) = 1$ is:

$$P(Y_1 = 1) = P(X = 0)(1 - \theta) + P(X \geq 1)\theta$$

$$P(Y_1 = 1) = (1/4)(1 - \theta) + (3/4)\theta$$

$$\boldsymbol{P(Y_1 = 1) = (1/2)\theta + (1/4)}$$

(b) Because this problem involves discrete random variable, we can model the likelihood with the same function as we model the probability. To compute the probability, there are some components that we need to define:

1. How many combination of events (value of y_i) are there, stated by $\binom{n}{\sum_{i=1}^{n} y_i}$. For example, if n=2, there are four possibilities: [0,0], [0,1], [1,0], [1,1].
2. For each event, if $y_i = 1$ the probability would be $P(y_i = 1)$ else if $y_i = 0$ it would be $P(y_i = 0)$. This can be stated as $P(y_i = 1)^{y_i} P(y_i = 0)^{1-y_i}$.

Thus, the likelihood is:

$$L(\theta|n, \sum_{i=1}^{n} y_i) = P(\sum_{i=1}^{n} y_i)|n, \theta)$$

$$L(\theta|n, \sum_{i=1}^{n} y_i) = \binom{n}{\sum_{i=1}^{n} y_i} \prod_{i=1}^{n} P(y_i = 1)^{y_i} P(y_i = 0)^{1-y_i}$$

$$L(\theta|n, \sum_{i=1}^{n} y_i) = \binom{n}{\sum_{i=1}^{n} y_i} \prod_{i=1}^{n} P(y_i = 1)^{y_i} (1 - P(y_i = 1))^{1-y_i}$$

$$L(\theta|\sum_{i=1}^{n} y_i) = \binom{n}{\sum_{i=1}^{n} y_i} \prod_{i=1}^{n} (\frac{1}{2}\theta + \frac{1}{4})^{y_i} (-\frac{1}{2}\theta + \frac{3}{4})^{1-y_i}$$

Then, the log likelihood would be:

$$\ln L(\theta|\sum_{i=1}^{n} y_i) = \ln \binom{n}{\sum_{i=1}^{n} y_i} \prod_{i=1}^{n} (\frac{1}{2}\theta + \frac{1}{4})^{y_i} (-\frac{1}{2}\theta + \frac{3}{4})^{1-y_i}$$

$$\ln L(\theta|\sum_{i=1}^{n} y_i) = \ln \binom{n}{\sum_{i=1}^{n} y_i} + \ln \prod_{i=1}^{n} (\frac{1}{2}\theta + \frac{1}{4})^{y_i} (-\frac{1}{2}\theta + \frac{3}{4})^{1-y_i}$$

$$\ln L(\theta|\sum_{i=1}^{n} y_i) = \ln \binom{n}{\sum_{i=1}^{n} y_i} + \sum_{i=1}^{n} \ln(\frac{1}{2}\theta + \frac{1}{4})^{y_i} + \sum_{i=1}^{n} \ln(-\frac{1}{2}\theta + \frac{3}{4})^{1-y_i}$$

$$\ln L(\theta|\sum_{i=1}^{n} y_i) = \ln \binom{n}{\sum_{i=1}^{n} y_i} + \sum_{i=1}^{n} y_i \ln(\frac{1}{2}\theta + \frac{1}{4}) + \sum_{i=1}^{n} (1 - y_i) \ln(-\frac{1}{2}\theta + \frac{3}{4})$$

$$\boldsymbol{\ln L(\theta|\sum_{i=1}^{n} y_i) = \ln \binom{n}{\sum_{i=1}^{n} y_i} + \sum_{i=1}^{n} y_i \ln(\frac{1}{2}\theta + \frac{1}{4}) + (n - \sum_{i=1}^{n} (y_i)) \ln(-\frac{1}{2}\theta + \frac{3}{4})}$$

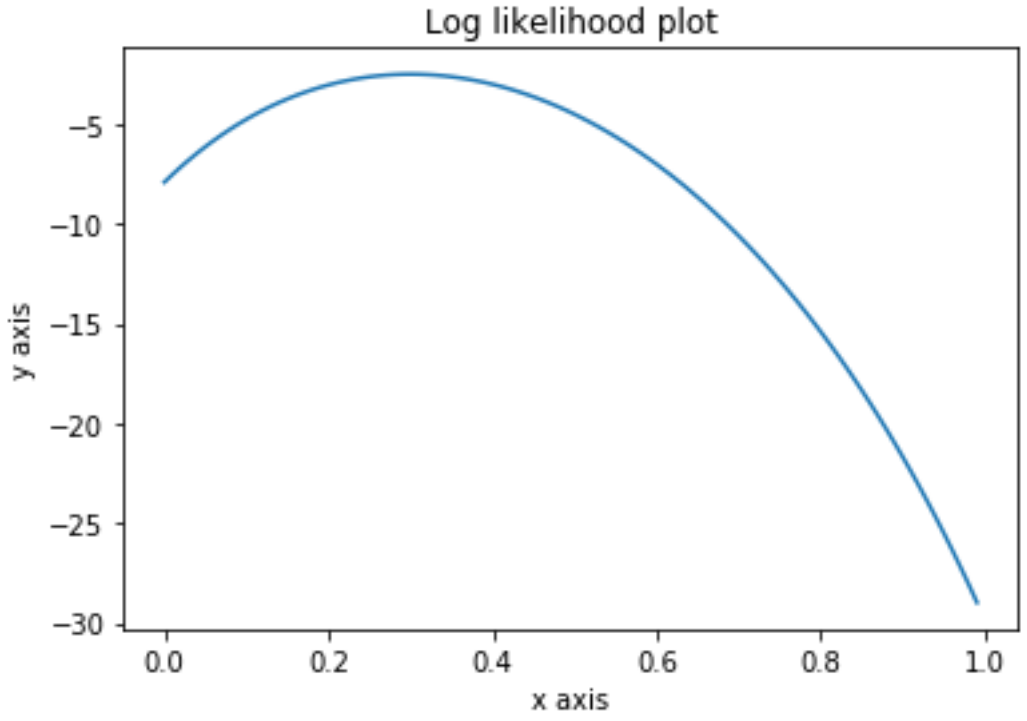

Figure 2: Log Likelihood Plot

(c) Above is the plot of log-likelihood as a function of $\theta \in [0, 1]$ given n=100 and $\sum_{i=1}^{n} y_i = 40$. The function that corresponds to the plot is derived below by substituting given values:

$$L(\theta|n = 100, \sum_{i=1}^{n} y_i = 40) = \ln \binom{100}{\sum_{i=1}^{100} y_i} + \sum_{i=1}^{100} y_i \ln(\frac{1}{2}\theta + \frac{1}{4}) + (100 - \sum_{i=1}^{100} (y_i)) \ln(-\frac{1}{2}\theta + \frac{3}{4})$$

$$L(\theta|n = 100, \sum_{i=1}^{n} y_i = 40) = \ln \binom{100}{40} + 40 \ln(\frac{1}{2}\theta + \frac{1}{4}) + (100 - 40) \ln(-\frac{1}{2}\theta + \frac{3}{4})$$

$$L(\theta|n=100, \sum_{i=1}^{n} y_i = 40) = \ln(\frac{100!}{60!40!}) + 40\ln(\frac{1}{2}\theta + \frac{1}{4}) + 60\ln(-\frac{1}{2}\theta + \frac{3}{4})$$

$$\boldsymbol{L(\theta|n=100, \sum_{i=1}^{n} y_i = 40) = 64.7906 + 40\ln(\frac{1}{2}\theta + \frac{1}{4}) + 60\ln(-\frac{1}{2}\theta + \frac{3}{4})}$$

To determine the θ with the highest likelihood, we can take the first derivative of $L(\theta)$ and then find θ so that it equals to zero.

$$\frac{d}{d\theta}L(\theta) = 0$$

$$\frac{d}{d\theta}\ln(\frac{100!}{60!40!}) + \frac{d}{d\theta}40\ln(\frac{1}{2}\theta + \frac{1}{4}) + \frac{d}{d\theta}60\ln(-\frac{1}{2}\theta + \frac{3}{4}) = 0$$

$$\frac{(40)(\frac{1}{2})}{\frac{1}{2}\theta + \frac{1}{4}} + \frac{(60)(-\frac{1}{2})}{-\frac{1}{2}\theta + \frac{3}{4}} = 0$$

$$\frac{20}{\frac{1}{2}\theta + \frac{1}{4}} + \frac{-30}{-\frac{1}{2}\theta + \frac{3}{4}} = 0$$

$$\frac{-10\theta + 15 - 15\theta - 7.5}{(1/2)\theta + 1/4)((-1/2)\theta + 3/4)} = 0$$

$$\frac{-25\theta + 7.5}{(1/2)\theta + 1/4)((-1/2)\theta + 3/4)} = 0$$

$$\theta = \frac{7.5}{25} = \frac{3}{10} = 0.3$$

$$\boldsymbol{\hat{\theta}_{mle} = 0.3}$$

Problem 7

(a) The author (1) discusses a named entity recognition problem for Arabic words. The problem is modeled as a classification task. Given a sequence of words, the model needs to predict an entity tag for every word in a sentence, in other words, a sequence labelling task. There are four defined tags, which are person, location, company and others. Machine learning, in this case, is used in a supervised manner. The labeled data is split into train and test data. The detail about the dataset can be found in section IV A.

(b) The iid model of data is not appropriate for this application because:

1. In the fourth paragraph of introduction section, the author explains about Optional Short Vowels problem in Arabic language. In short, for some consonants, the writer can omit the vowels so that there can be more than one interpretation. In the example, the author shows an Arabic word that can either be a person or organization depending on the context. If the previous word is a person, that word is more likely a person and vice versa if the previous word is an organization. Thus, this problem doesn't comply to the independent random variables principle as the previous random variable(s) may affect the probability of a currently observed random variable.
2. There might be new words in the test data that haven't been observed before, e.g. words that are not in the defined dictionary. This is mentioned by the author in the abstract and also in the eight paragraph of the introduction. Thus, the probability of the data points (in this case the words) in the training and testing data might be different, meaning that they are not identically distributed.

(c) Yes. The author uses a neural network architecture called LSTM that can capture long term dependencies within a sequence. The author also uses a so-called self-attention layer, a mechanism to compute which word in the sequence contributes the most to the probability of a tag that is being observed. For example, when predicting the third tag, we might want to give more weight on the second tag rather than the first tag. This is nicely explained in Section III D.

References

[1] M. N. A. Ali, G. Tan, and A. Hussain, "Boosting arabic named-entity recognition with multi-attention layer," *IEEE Access*, vol. 7, pp. 46575–46582, 2019. https://ieeexplore.ieee.org/document/8685084.