25T1 9417 HW1 Solutions

z5270589 YiJie Zhao

Question1

(a)

Create avariable X containing only features, and a variably y containing the target (Heart Disease). Remove the Last Checkup feature:

- We use X.info() and y.info() to show the results respectively, we can see that y only contains the target-Heart Disease and X has all except the Last Check up and Heart Disease.

y:

```
Series name: Heart_Disease
Non-Null Count  Dtype
--------------  -----
100 non-null    float64
dtypes: float64(1)
```

x:

```
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             93 non-null     float64
 1   Gender          100 non-null    object
 2   Height_feet     100 non-null    float64
 3   Weight_kg       100 non-null    int64
 4   Blood_Pressure  100 non-null    object
 5   Cholesterol     100 non-null    int64
 6   Smoker          92 non-null     object
dtypes: float64(2), int64(2), object(3)
```

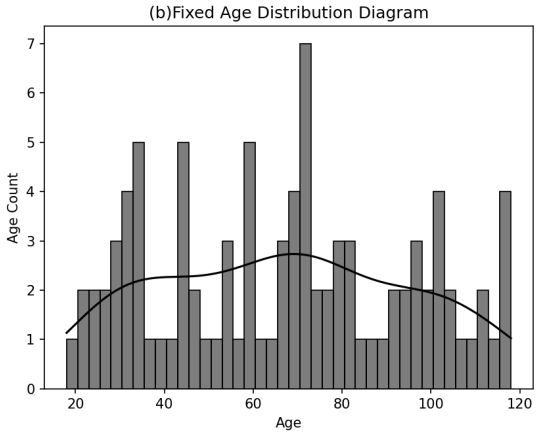
(b)

For Age, some values are negative. You are informed that this is a data entry error and negatives should be replaced with their positive versions. That is, −x should be replaced with x:

- We use X.describe() to attain the min value for age, we can see that the value is 18, which is not negative.

```
              Age  Height_feet   Weight_kg  Cholesterol
count   93.000000   100.000000  100.000000   100.000000
mean    66.989247     5.632070   89.760000   228.950000
std     28.076562     0.440279   29.334752    29.262785
min     18.000000     4.920000  -70.000000   180.000000
25%     44.000000     5.379200   71.750000   200.750000
50%     69.000000     5.641600   90.500000   231.500000
75%     91.000000     5.936800  113.000000   253.500000
max    118.000000     8.231000  129.000000   280.000000
```

- Meanwhile we use a histogram to shows the distribution of current age, we can find that they are all positive values now:



(c)

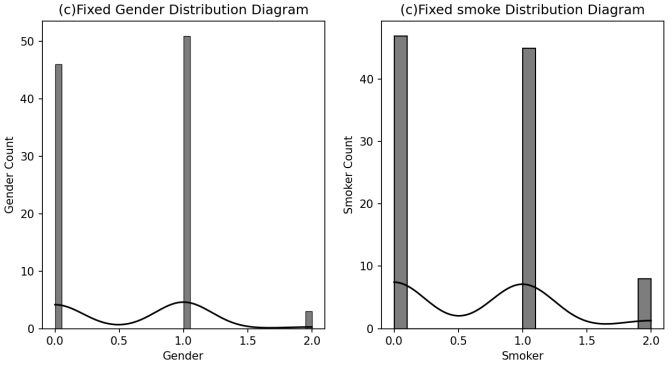
For Gender and Smoker, the variables have been coded in inconsistent ways. For example, Female gender is encoded as 'Female' and as 'F'. Write code to make these codings consistent. Then use categorical encoding instead. For gender, map (Male/M,Female/F, Unknown) to (0,1,2). For Smoker, map (No/N,Yes/Y,Nan) to (0,1,2):

- We use the print(X) to check the status of current Gender and Smoker, we can see that all values of gender and smoker has been transferred to 0/1/2:

	Age	Gender	Height_feet	Weight_kg	Blood_Pressure	Cholesterol	Smoker
0	45.0	0	5.0512	61	120/80	207	0
1	101.0	1	4.9856	78	130/85	208	0
2	NaN	0	5.3792	51	125/75	267	1
3	58.0	0	5.2152	126	138/88	221	0
4	64.0	1	4.9200	102	142/90	182	0
5	50.0	0	6.1336	75	115/70	245	2
6	69.0	2	5.6416	67	128/82	274	1
7	31.0	0	5.0840	116	135/78	256	1
8	70.0	1	5.3792	95	140/85	250	1
9	31.0	1	5.9696	118	122/76	191	1
10	69.0	1	5.7728	125	130/80	236	1
11	21.0	1	6.0352	109	125/78	209	0
12	91.0	0	5.9368	115	136/84	222	0
13	80.0	1	6.0352	68	145/92	260	1
14	71.0	0	5.8384	63	118/74	257	0
15	85.0	1	6.1008	72	133/87	180	0
16	NaN	1	5.3136	86	124/79	194	1
17	34.0	1	5.7728	84	129/81	182	1
18	35.0	1	5.3464	84	138/86	247	0
19	73.0	1	5.8056	113	127/77	193	2
20	29.0	0	5.0512	112	132/83	222	0
21	104.0	0	5.8056	101	144/91	235	1
22	87.0	0	5.8056	113	121/75	192	1
23	NaN	0	6.1336	91	126/79	253	0
24	NaN	1	5.1496	82	135/82	248	1
25	52.0	0	5.6416	64	139/88	221	1
26	71.0	1	5.4776	114	118/72	261	0
27	NaN	0	5.1824	111	134/86	259	1

-

- The two histograms also shows the current Smoker and gender distribution:



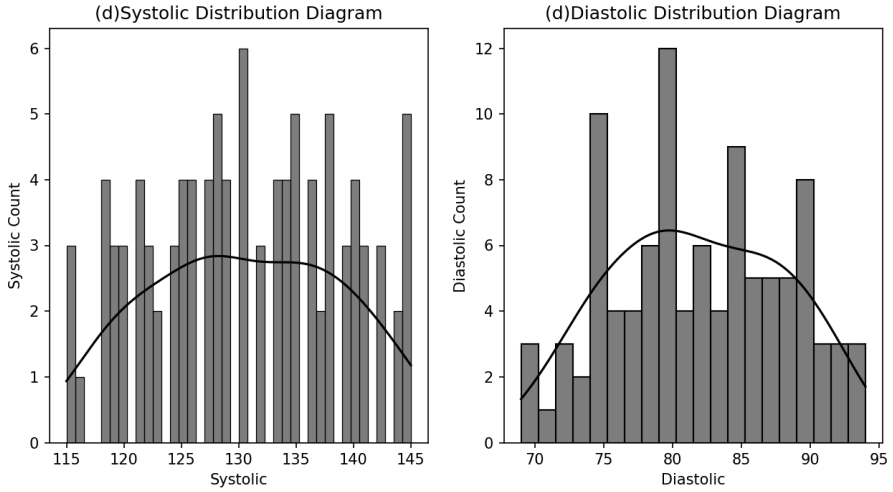
(d)

Blood pressure is given in the form systolic/diastolic. Write code to create two variables, systolic and diastolic. Remove the original blood pressure variable:

- We first use X.info() to show these two new term:

```
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          93 non-null     float64
 1   Gender       100 non-null    int64
 2   Height_feet  100 non-null    float64
 3   Weight_kg    100 non-null    int64
 4   Cholesterol  100 non-null    int64
 5   Smoker       100 non-null    int64
 6   Systolic     100 non-null    float64
 7   Diastolic    100 non-null    float64
```

- The two histograms can show the new two variables systolic and diastolic:



(e)

Using sklearn.model selection.train sets. Set the test test split, split the data into training and test size parameter to 0.3, and the random state to '2' for reproducibility:

- This is the function we used here:
-
```
X_trainingSet, X_testSet, y_trainingSet, y_testSet = train_test_split(X, y, test_size=0.3, random_state=2)
```
- We use y/X_trainingSet.info(), y/X_testSet.info() to show the new **four** datasets:

- For X training set:

```
(e)The information of X training Set :
<class 'pandas.core.frame.DataFrame'>
Index: 70 entries, 65 to 40
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          70 non-null     float64
 1   Gender       70 non-null     int64
 2   Height_feet  70 non-null     float64
 3   Weight_kg    70 non-null     int64
 4   Cholesterol  70 non-null     int64
 5   Smoker       70 non-null     int64
 6   Systolic     70 non-null     float64
 7   Diastolic    70 non-null     float64
dtypes: float64(4), int64(4)
```

- For X test set:

```
(e)The information of X test Set :
<class 'pandas.core.frame.DataFrame'>
Index: 30 entries, 83 to 62
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          23 non-null     float64
 1   Gender       30 non-null     int64
 2   Height_feet  30 non-null     float64
 3   Weight_kg    30 non-null     int64
 4   Cholesterol  30 non-null     int64
 5   Smoker       30 non-null     int64
 6   Systolic     30 non-null     float64
 7   Diastolic    30 non-null     float64
dtypes: float64(4), int64(4)
```

- For y training set:

```
(e)The information of y training Set :
<class 'pandas.core.series.Series'>
Index: 70 entries, 65 to 40
Series name: Heart_Disease
Non-Null Count  Dtype
--------------  -----
70 non-null     float64
```

- For y test set:

```
(e)The information of y test Set :
<class 'pandas.core.series.Series'>
Index: 30 entries, 83 to 62
Series name: Heart_Disease
Non-Null Count  Dtype
--------------  -----
30 non-null     float64
dtypes: float64(1)
```

(f)

Now, note that somevalues in 'Age' in your test data are missing. We will manually impute values according to the following rule: If a Male (Female) is missing their age, set it to be the median of all other Male (Female) patients. Note that you should NOT use test data for this step, so your medians should be computed based on training set data. Be careful not to include missing values when calculating your median:

- From (e), we notice that some age value in X testSet is missing(30 miss 7), after (f), we can see that all ages in X testSet has the value(30 non-null):

```
(e)The information of X test Set :
<class 'pandas.core.frame.DataFrame'>
Index: 30 entries, 83 to 62
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          23 non-null     float64
 1   Gender       30 non-null     int64
 2   Height_feet  30 non-null     float64
 3   Weight_kg    30 non-null     int64
 4   Cholesterol  30 non-null     int64
 5   Smoker       30 non-null     int64
 6   Systolic     30 non-null     float64
 7   Diastolic    30 non-null     float64
dtypes: float64(4), int64(4)
```

```
(f)The information of X test Set :
<class 'pandas.core.frame.DataFrame'>
Index: 30 entries, 83 to 62
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          30 non-null     float64
 1   Gender       30 non-null     int64
 2   Height_feet  30 non-null     float64
 3   Weight_kg    30 non-null     int64
 4   Cholesterol  30 non-null     int64
 5   Smoker       30 non-null     int64
 6   Systolic     30 non-null     float64
 7   Diastolic    30 non-null     float64
dtypes: float64(4), int64(4)
```

- And we also show the comparison of previous miss age num and current:

```
print("\033[1mNumbers of 'Age' values missing in my original X data: \033[0m", X['Age'].isna().sum())
print("\033[1mNumbers of 'Age' values missing in my test data: \033[0m", X_trainingSet['Age'].isna().sum())
```

```
Numbers of 'Age' values missing in my original X data:  7
Numbers of 'Age' values missing in my test data:  0
```

(g)

Scale the columns: 'Age', 'Height feet', 'Weight kg', 'Cholesterol', 'Systolic', 'Diastolic' using a min-max normalizer. This means that for each feature, you should replace x with $\frac{x-\min}{\max-\min}$, where min,max are the minimum and maximum values for that feature column, respectively. Make sure to do this separately for train and test data:

- We record and compare the original X trainingSet information with current fixed X trainingSet information:

Original X_Training set before using a min-max normalizer

	Age	Gender	Height_feet	Weight_kg	Cholesterol	Smoker	Systolic	Diastolic
count	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000
mean	67.485714	0.657143	5.632671	90.542857	228.600000	0.571429	130.071429	81.257143
std	28.246197	0.561719	0.459152	31.029059	29.986664	0.603647	8.436157	6.376328
min	18.000000	0.000000	4.920000	-70.000000	180.000000	0.000000	115.000000	69.000000
25%	41.750000	0.000000	5.412000	73.000000	198.750000	0.000000	122.250000	76.000000
50%	69.500000	1.000000	5.641600	94.000000	232.000000	1.000000	130.000000	81.000000
75%	90.500000	1.000000	5.830200	113.750000	254.500000	1.000000	136.750000	86.000000
max	118.000000	2.000000	8.231000	129.000000	280.000000	2.000000	145.000000	93.000000

Current X_Training set after using a min-max normalizer

	Age	Gender	Height_feet	Weight_kg	Cholesterol	Smoker	Systolic	Diastolic
count	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000	70.000000
mean	0.494857	0.657143	0.215244	0.806748	0.486000	0.571429	0.502381	0.510714
std	0.282462	0.561719	0.138675	0.155925	0.299867	0.603647	0.281205	0.265680
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.237500	0.000000	0.148596	0.718593	0.187500	0.000000	0.241667	0.291667
50%	0.515000	1.000000	0.217940	0.824121	0.520000	1.000000	0.500000	0.500000
75%	0.725000	1.000000	0.274902	0.923367	0.745000	1.000000	0.725000	0.708333
max	1.000000	2.000000	1.000000	1.000000	1.000000	2.000000	1.000000	1.000000

- Since we also do the same fit_transform for X testSet, it expects the same performance.

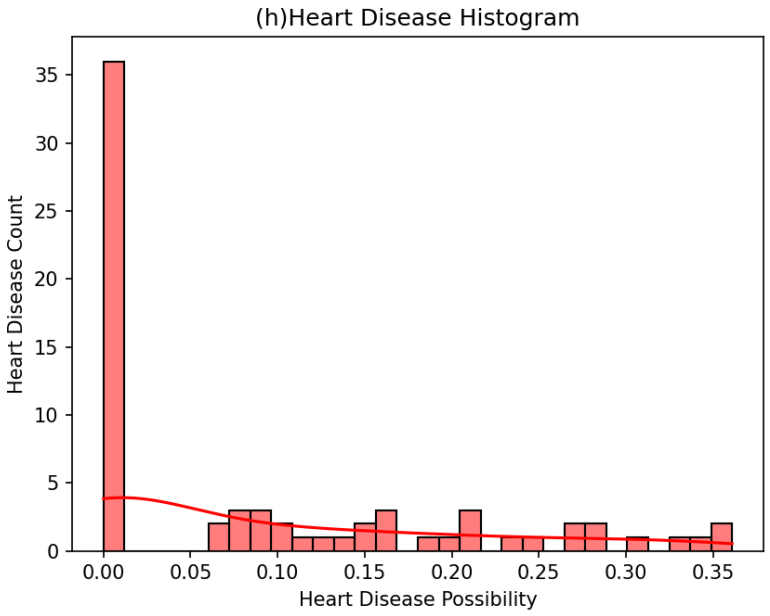
```
X_trainingSet[Target_Scale_Columns] = X_min_max_normalizer.fit_transform(X_trainingSet[Target_Scale_Columns])
# For X test Set, we do also use fit_transform here to make sure to do this separately for train and test data.
X_testSet[Target_Scale_Columns] = X_min_max_normalizer.fit_transform(X_testSet[Target_Scale_Columns])
```

-

(h)

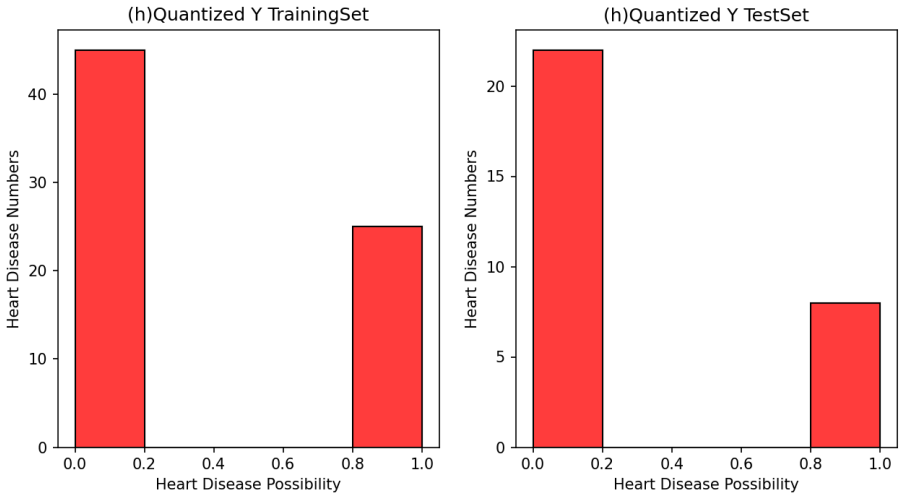
Plot a histogram of your target variable (from your training data). You should notice that a large portion of the target value is clustered around zero. Do you think linear regression is a reasonable model for this data? Create a new target variable by quantizing the original target variable. You can do this by setting values below a certain threshold (say 0.1) to be 0 and those above the threshold to be 1:

- This is the histogram of your target variable (from my training data):



-

- Histograms for the Quantized Y TrainingSet and Quantized Y TestSet:



-

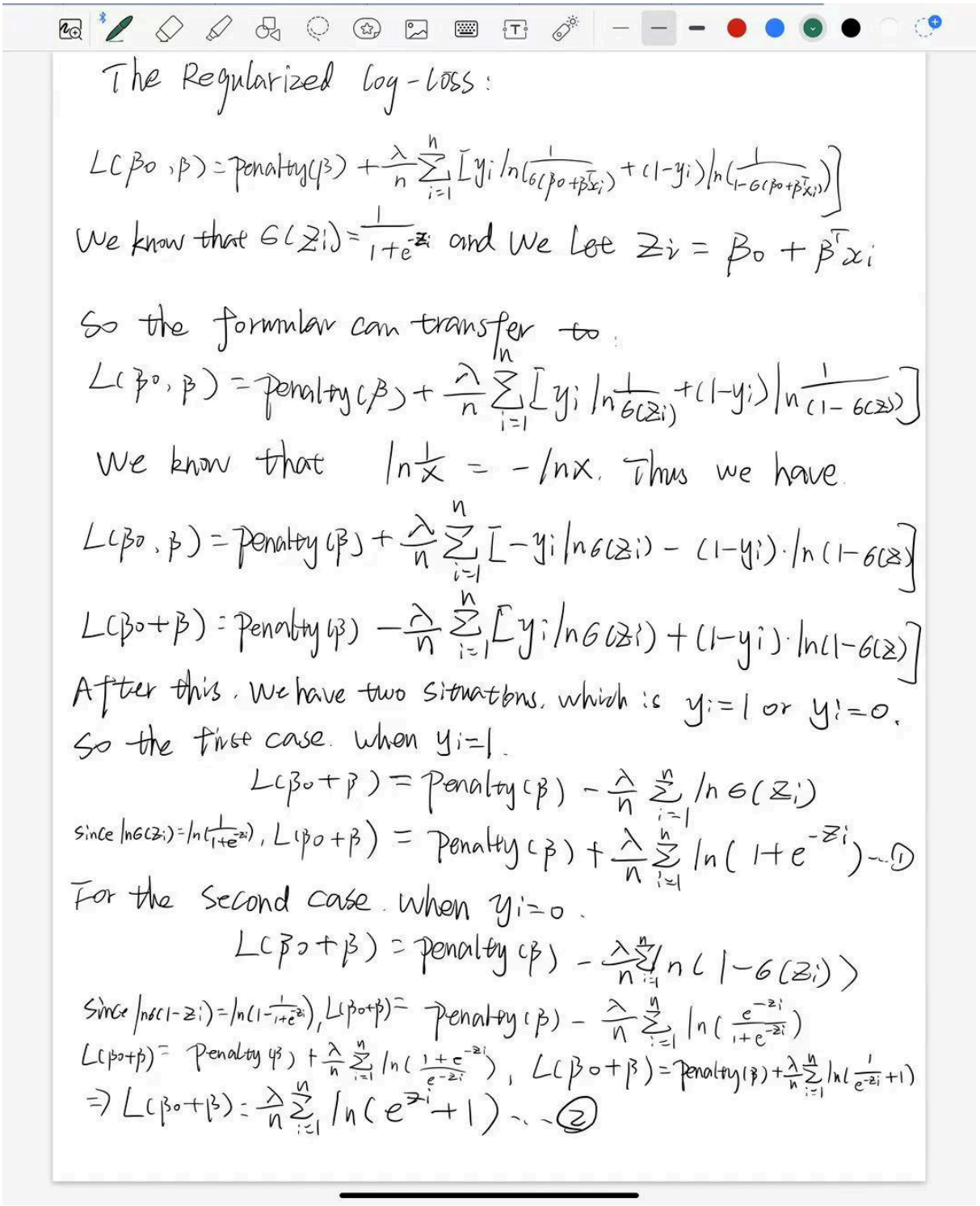
- From the lecture slides, we can learn that the assumptions of linear regression contains the featureNormality of residuals, which for any fixed value of y should be normally distributed, or we can say the target value y should follow a normal distribution,but from the diagram above we can notice that a large portion of the target value is clustered around 0 and others around 1, so I believe the method linear regression is not a reasonable model for this data.Since this likes a clarification problem, we can Quantized y to binary valued and apply the Logistic Regression model to achieve a better outcome.

Question2

(a)

Argue rigorously that the two objectives are identical:

- I did this by hand draft, my working is shown as the **two** pictures below:



The Regularized log-loss:

$$L(\beta_0,\beta) = \text{Penalty}(\beta) + \frac{\lambda}{n}\sum_{i=1}^{n}\left[y_i \ln\left(\frac{1}{G(\beta_0+\beta^T x_i)}\right) + (1-y_i)\ln\left(\frac{1}{1-G(\beta_0+\beta^T x_i)}\right)\right]$$

We know that $G(z_i) = \frac{1}{1+e^{-z_i}}$ and we Let $z_i = \beta_0 + \beta^T x_i$

So the formular can transfer to:

$$L(\beta_0,\beta) = \text{Penalty}(\beta) + \frac{\lambda}{n}\sum_{i=1}^{n}\left[y_i \ln\frac{1}{G(z_i)} + (1-y_i)\ln\frac{1}{(1-G(z_i))}\right]$$

We know that $\ln\frac{1}{x} = -\ln x$. Thus we have

$$L(\beta_0,\beta) = \text{Penalty}(\beta) + \frac{\lambda}{n}\sum_{i=1}^{n}\left[-y_i \ln G(z_i) - (1-y_i)\cdot\ln(1-G(z))\right]$$

$$L(\beta_0+\beta) = \text{Penalty}(\beta) - \frac{\lambda}{n}\sum_{i=1}^{n}\left[y_i \ln G(z_i) + (1-y_i)\cdot\ln(1-G(z))\right]$$

After this, we have two situations, which is $y_i=1$ or $y_i=0$.

So the first case when $y_i=1$.

$$L(\beta_0+\beta) = \text{Penalty}(\beta) - \frac{\lambda}{n}\sum_{i=1}^{n}\ln G(z_i)$$

since $\ln G(z_i) = \ln\left(\frac{1}{1+e^{-z_i}}\right)$, $L(\beta_0+\beta) = \text{Penalty}(\beta) + \frac{\lambda}{n}\sum_{i=1}^{n}\ln(1+e^{-z_i})$ ... ①

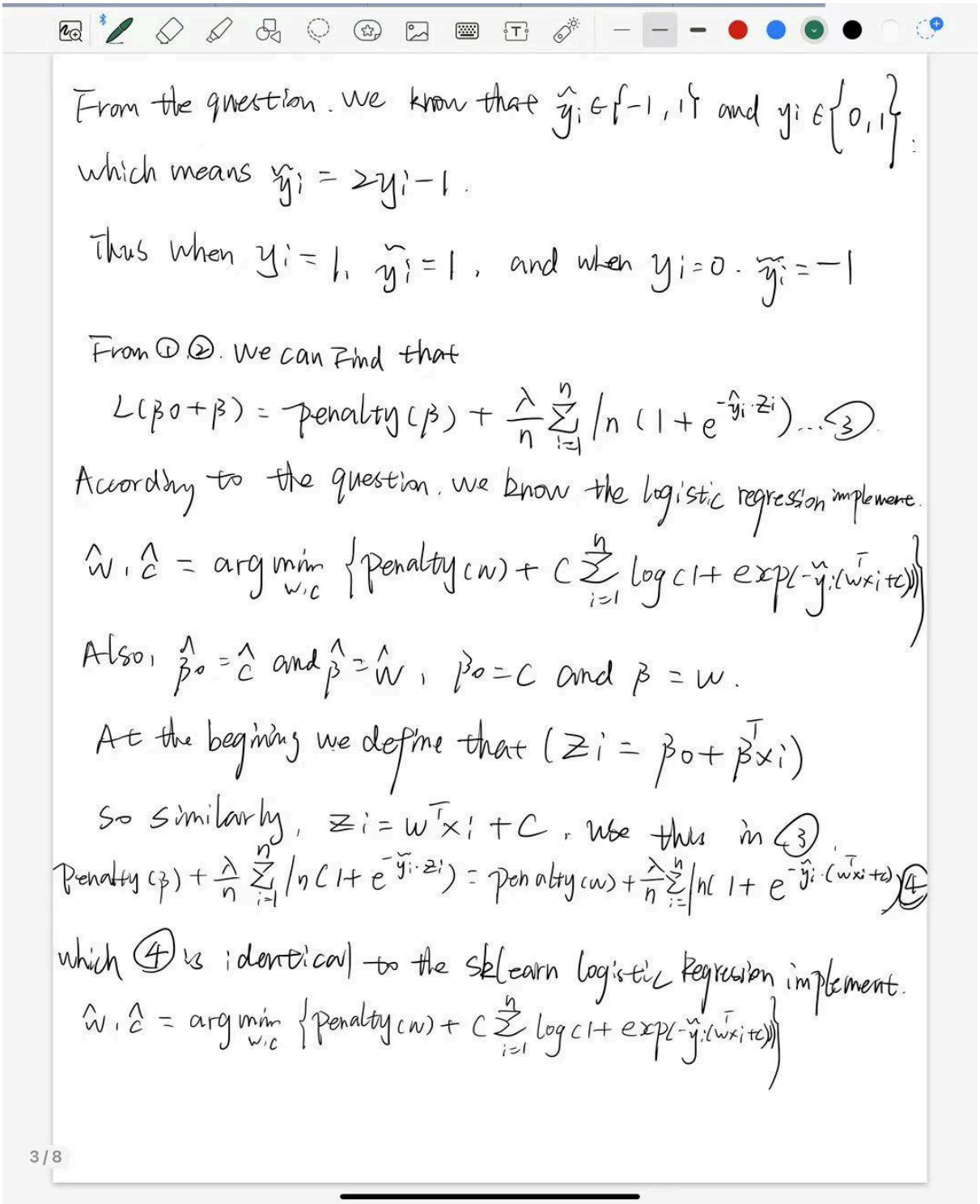
For the second case when $y_i=0$.

$$L(\beta_0+\beta) = \text{Penalty}(\beta) - \frac{\lambda}{n}\sum_{i=1}^{n}\ln(1-G(z_i))$$

Since $\ln(1-z_i) = \ln\left(1-\frac{1}{1+e^{-z_i}}\right)$, $L(\beta_0+\beta) = \text{Penalty}(\beta) - \frac{\lambda}{n}\sum_{i=1}^{n}\ln\left(\frac{e^{-z_i}}{1+e^{-z_i}}\right)$

$L(\beta_0+\beta) = \text{Penalty}(\beta) + \frac{\lambda}{n}\sum_{i=1}^{n}\ln\left(\frac{1+e^{-z_i}}{e^{-z_i}}\right)$, $L(\beta_0+\beta) = \text{Penalty}(\beta) + \frac{\lambda}{n}\sum_{i=1}^{n}\ln\left(\frac{1}{e^{-z_i}}+1\right)$

$\Rightarrow L(\beta_0+\beta) = \frac{\lambda}{n}\sum_{i=1}^{n}\ln(e^{z_i}+1)$ ... ②



Further, describe the role of C in the objectives, how does it compare to the standard Ridge parameter λ as you have seen in the class?

- C in the objectives can control the weight of the classification error term in the loss function, which means if the model is overfitting, we can decrease C to enhance the regularization, if the model is underfitting, then we can increase C to reduce the regularization.

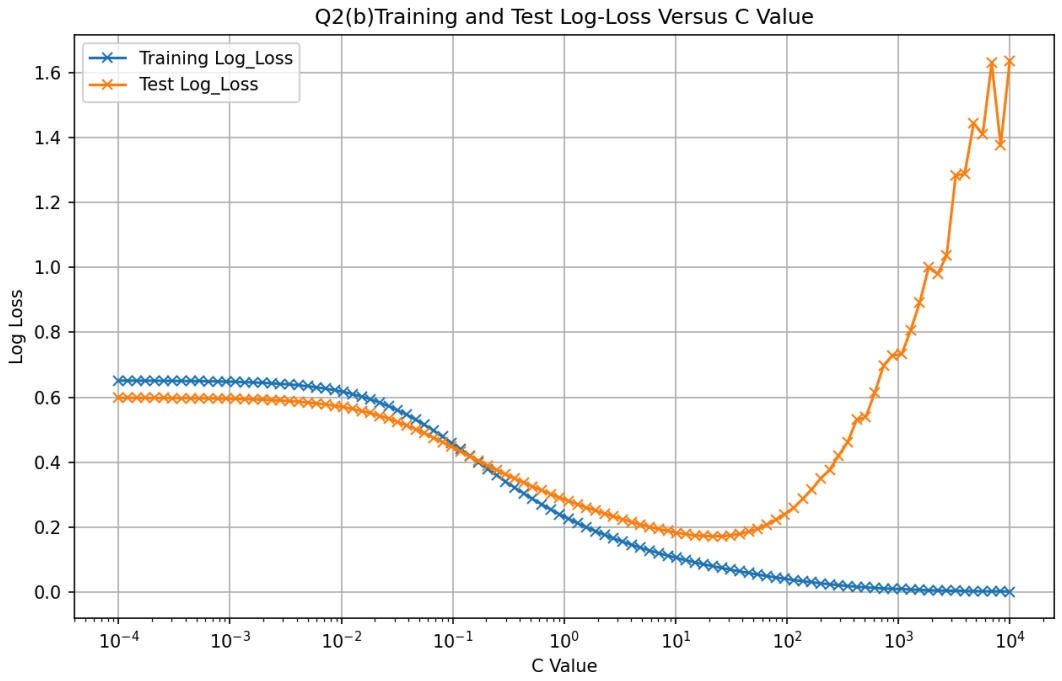
- As for C compared to the standard Ridge parameter λ as you have seen in the class, in the class.

$$\theta^* = \arg\min_{\theta}(y - X\theta)^T(y - X\theta) + \lambda\|\theta\|^2$$

Different from the λ in question2(a), this λ is in the penalty term. Since we need to increase λ to enhance the regularization and decrease C to enhance the regularization, we can conclude that C and standard Ridge parameter λ are inversely related, which C = 1/λ.

(b)

The shape of the **two** loss curves are shown below:



The X-axis of the plot represents the value of C. As the value of C increases, the regularization strength decreases, which can lead to the model becoming more prone to overfitting.
The Y axis of the plot represents the log-loss value, the smaller the log-loss is, the better model we will achieve.

How would you pick C based on these plots? State your choice of C.

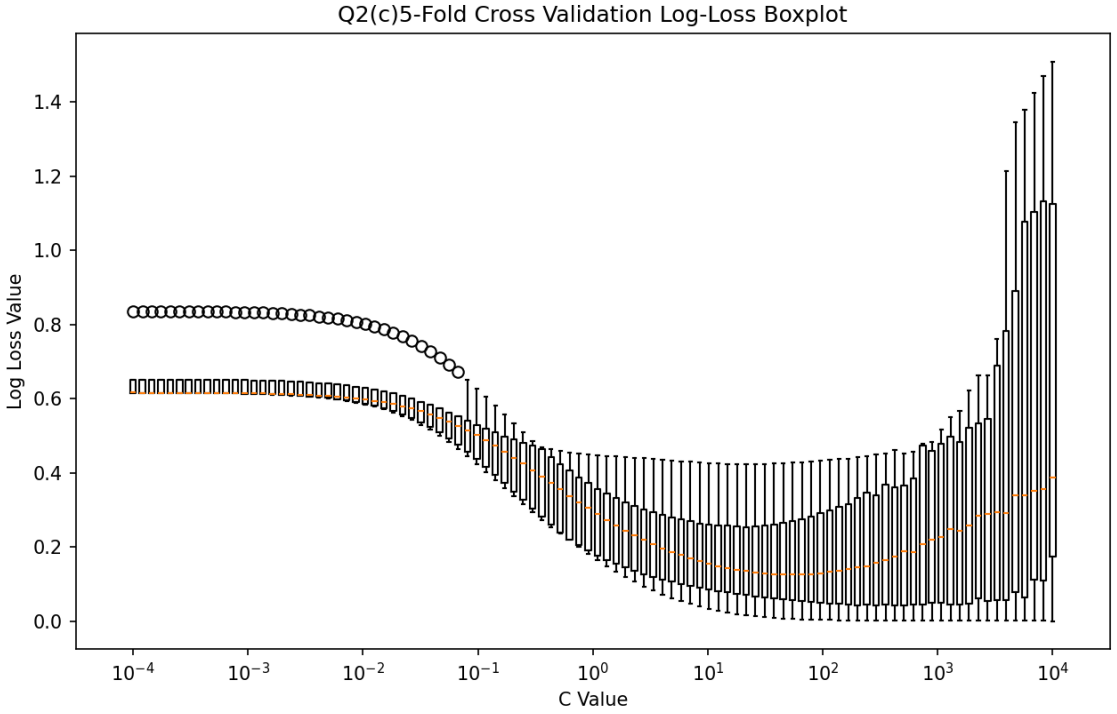
- For the curve of **Training Log-Loss**, we can notice that when C is small, the log-loss is large and can not capture the characteristics or patterns of data well, and the model seems underfitting. When the value of C increases, the regularization strength decreases, leading to the log-loss decreasing and the model becomes better in the training set. Until C reaches 10^4, we get the minimized log-loss value.

- For the curve of **Test Log-Loss**, we can notice that when C is small, the log-loss is large and can not capture the characteristics or patterns of data well, and the model seems underfitting. When the value of C increases, the regularization strength decreases and leads to the log-loss decreasing. When C values between 10^1 — 10^2(about 30-40 in the diagram), we get the minimized log-loss value; if C values still increase, we can see that the model will become more overfitting, the log-loss value will increase.

- When picking C, we should more focus on the Test Log-Loss, so we can conclude that we should pick the C value that can lead to the minimized Log-Loss value of the Test dataset, which in this case we can get the best model that has the best generalization strength, from the diagram, we know that when C value between 10^1 — 10^2(about 30-40) gives us the minimized Log-Loss value of the Test dataset, so we should pick the C value between 10^1 — 10^2(around 30-40).

(c)

For each C, plot a box-plot over the 5 CV scores:



When we plot the boxplot, to make the plot look better, we turn the C value to ln(c) to prevent most of the values from staying on the left side, so we can see that now the value is equally distributed in the X axis.

```
plt.boxplot(Test_fold_log_losses, positions=np.log10(C_100_values), widths=0.05)
```

Report the value of C that gives you the best CV performance in terms of log-loss:

- To choose the best C value that brings me the best CV performance in terms of log-loss, firstly we calculate the mean log-loss for each C value across the 5 folds

validation:

```
mean_log_losses = [np.mean(log_loss) for log_loss in Test_fold_log_losses]
```

After that we can select the minimized mean log loss corresponding index among 100 5 folds mean log loss values:

```
best_C_index = np.argmin(mean_log_losses)
```

After that we can find the best C value according to its index(leads to the smallest mean log-loss value):

```
Best_C = C_100_values[best_C_index]
```

And the best C value is `Q2(c)Best C value: 37.649358067924716`.

Re-fit the model with this chosen C, and report both train and test accuracy using this model:

- After re-fit the model.

```
# Re-fit the model with this chosen C:
model = LogisticRegression(C=Best_C, penalty='l2', solver='lbfgs', max_iter=200)
model.fit(X_TrainingSet_fold, y_TrainingSet_quantizing_fold)

best_y_train_pred = model.predict(X_trainingSet)
best_y_test_pred = model.predict(X_testSet)
```

We can obtain the train and test accuracy by apply the accuracy_score() function:

```
training_accuracy = accuracy_score(y_trainingSet_quantizing, best_y_train_pred)
print(f"\033[1mQ2(c)The Training Accuracy is: {training_accuracy:.4f}\033[0m")

test_accuracy = accuracy_score(y_testSet_quantizing, best_y_test_pred)
print(f"\033[1mQ2(c)The Test Accuracy is: {test_accuracy:.4f}\033[0m")
```

Which the **train accuray** is: `Q2(c)The Training Accuracy is: 1.0000`.

The test accuracy is: `Q2(c)The Test Accuracy is: 0.9333`.

A screenshot of my code for this section(c):

```python
# (c)
# Test_fold_log_losses to record all log-loss value.
Test_fold_log_losses = []
# We will split the train data into 5 folds.
fold_num = 5

for C in C_100_values:
    # iter_Test_fold_log_losses used to record the log-loss (5 folds)of each C value
    iter_Test_fold_log_losses = []

    # 5 iterations, each part will be treated as the testset once.
    for i in range(fold_num):

        # begin_index and terminate_index used as the pointers to find the current train and test set.
        begin_index = i * (len(X_trainingSet) // fold_num)
        terminate_index = (i + 1) * len(X_trainingSet) // fold_num if i < (fold_num - 1) else len(X_trainingSet)

        # First get the X and y test set.
        X_TestSet_fold = X_trainingSet.iloc[begin_index:terminate_index]
        y_TestSet_quantizing_fold = y_trainingSet_quantizing.iloc[begin_index:terminate_index]

        # Then the X and y training set.
        X_TrainingSet_fold = pd.concat([X_trainingSet.iloc[:begin_index], X_trainingSet.iloc[terminate_index:]])
        y_TrainingSet_quantizing_fold = pd.concat([y_trainingSet_quantizing.iloc[:begin_index], y_trainingSet_quantizing.iloc[terminate_index:]])

        # fit a logistic regression model (using the LogisticRegression class in sklearn) on the splited training data
        model = LogisticRegression(C=C, penalty='l2', solver='lbfgs', max_iter=200)
        model.fit(X_TrainingSet_fold, y_TrainingSet_quantizing_fold)

        # use predict proba to generate predictions from your fitted models to plug into the log-loss
        y_test_fold_proba = model.predict_proba(X_TestSet_fold)[:, 1]

        # get the log loss value
        test_fold_log_losses = log_loss(y_TestSet_quantizing_fold, y_test_fold_proba)
        iter_Test_fold_log_losses.append(test_fold_log_losses)

    # Record the different 5 training set log-loss value
    Test_fold_log_losses.append(iter_Test_fold_log_losses)

# we should match the Log-Loss valuae with its corresponding C value
plt.figure(figsize=(10, 6))

# plot a box-plot over the 5 CV scores.
# we use np.log10(C_100_values) here to prevent from most of values in the left side, now it will distribute average in the x axis.
plt.boxplot(Test_fold_log_losses, positions=np.log10(C_100_values), widths=0.05)

# Since we turn C into lnC before, we need re-define the X axsis to make the box plot looks better.
Refine_X = np.arange(-4, 5, 1)
# We choose 9 keypoints in the X axis, which is 10^-4-10^4
plt.xticks(Refine_X, [f"$10^{{{x_scale}}}$" for x_scale in Refine_X])
plt.xlabel('C Value')
plt.ylabel('Log Loss Value')
plt.title('Q2(c)5-Fold Cross Validation Log-Loss Boxplot')
plt.show()
```

```python
# (c)
# Report the value of C that gives you the best CV performance in terms of log-loss:
# We calculate the mean log-loss for each(100) iter_Test_fold_log_losses
mean_log_losses = [np.mean(log_loss) for log_loss in Test_fold_log_losses]
# Then we can select the minimized mean log loss index among 100 5 folds mean log loss values.
best_C_index = np.argmin(mean_log_losses)
# This index is the C index that leads to the minimized mean mean log loss in the 5 folds, we can recognized it as the best C
Best_C = C_100_values[best_C_index]
print(f"\033[1mQ2(c)Best C value: {Best_C}\033[0m")

# Re-fit the model with this chosen C:
model = LogisticRegression(C=Best_C, penalty='l2', solver='lbfgs', max_iter=200)
model.fit(X_TrainingSet_fold, y_TrainingSet_quantizing_fold)

best_y_train_pred = model.predict(X_trainingSet)
best_y_test_pred = model.predict(X_testSet)

# Report both train and test accuracy using this model
# we use the function accuracy_score() to achieve that:
training_accuracy = accuracy_score(y_trainingSet_quantizing, best_y_train_pred)
print(f"\033[1mQ2(c)The Training Accuracy is: {training_accuracy:.4f}\033[0m")

test_accuracy = accuracy_score(y_testSet_quantizing, best_y_test_pred)
print(f"\033[1mQ2(c)The Test Accuracy is: {test_accuracy:.4f}\033[0m")
```

(d)

Provide Two Reasons for why this is the case:

We can find that after we applied the sklearn implementation of gridsearch, we get the new best C value is `Q2(d) Default Best C value: 7.054802310718645`, which is different from the best C value in (c) — 37.649.

Here is the **two reasons** that I believe this case happened:

- The **first** reason. In (C), we manually split the X trainingSet into 5 folds based on the original data index, but when we use the GridSearchCV in part (d). (shuffle=True) is the default, which means the X training set is randomly shuffled before being divided by kfold(), and the fold data will be more uniform and random than cross validation from scratch, this can reduce the impact of the bias of the order of the data. This difference will lead to the different best C value.
- The **second** reason. When selecting the Best C value. In (C), we use the value of log-loss as the criterion for judging best C value, but in (d), if we do not explicitly specify the scoring parameter, GridSearchCV will use scoring='accuracy' by default, which means GridSearchCV uses accuracy as the criterion for judging best C value, and this leads to the different best C value compared to (C).

Re-run the code with some changes to give consistent results to those we computed by hand:

- We solve the two reasons we mentioned above respectively.
- **Firstly** we use shuffle = False in the KFold function to achieve the same effect as manually splitting the X trainingSet into 5 folds based on the original data index.

```
cv_close_shuffle = KFold(n_splits=5, shuffle=False)
```

- **Secondly** we use 'neg_log_loss' as the standard for model evaluation, the reason we use the negative here is because 'neg_log_loss' is a standard scoring metric built in GridSearchCV, and in the Scoring criteria, the larger value means the better choice, but log-loss is the smaller the best, so we use -log-loss 'neg_log_loss' here to fit the scoring principle.

```
grid_lr = GridSearchCV(estimator=LogisticRegression(penalty='l2', solver='lbfgs'), cv=cv_close_shuffle, param_grid=param_grid, scoring='neg_log_loss')
```

Finally, we get the consistent results(Best C value) to those we computed by hand, which is about 37.649:

```
Q2(d) After Re-run Best C value: 37.649358067924716
Q2(d) After Re-run Best log-loss value: 0.17624069469182177
```

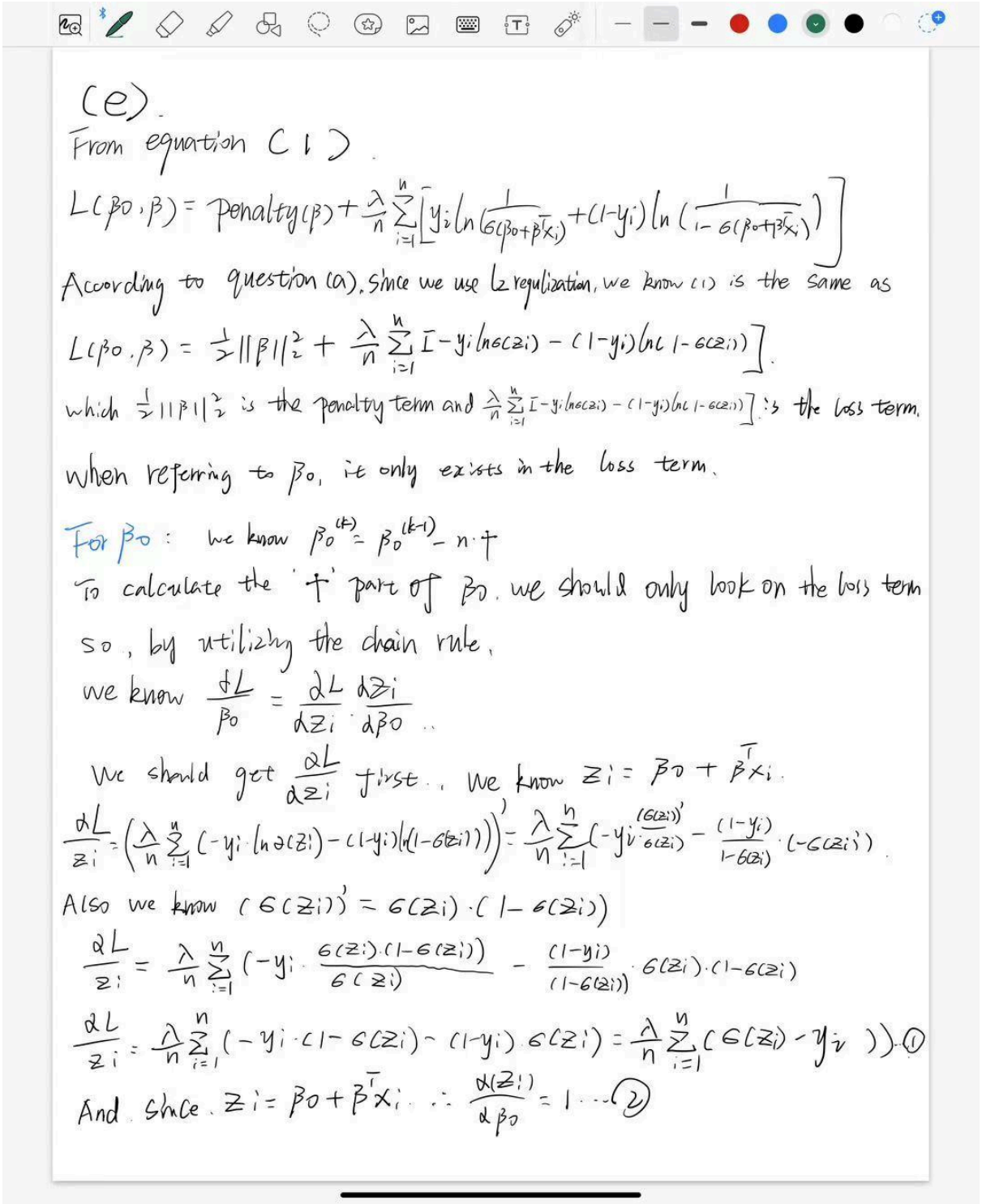
A screenshot of your code for this section(d):

```
# (d)
param_grid = {'C': C_100_values}
grid_lr = GridSearchCV(estimator=LogisticRegression(penalty='l2', solver='lbfgs'),cv=5, param_grid=param_grid)
grid_lr.fit(X_trainingSet, y_trainingSet_quantizing)

# (d)
# get the best C using gridsearch. We can find that the C value is different from (c)-37.649358067924716.
best_C = grid_lr.best_params_['C']
print(f"\033[1mQ2(d) Default Best C value: {best_C}\033[0m")

# (d)
# Re-run the code with some changes to give consistent results to those we computed by hand.

# Here we use shuffle = False to achieve the same effect as manually split the X trainingSet into 5 folds based on the original data index.
cv_close_shuffle = KFold(n_splits=5, shuffle=False)

# 'neg_log_loss' is a standard scoring metric built in GridSearchCV, the negative is because in the Scoring criteria, the larger value means the better choice, b
# so we use -log-loss 'neg_log_loss' here to fit the scoring principle.
grid_lr = GridSearchCV(estimator=LogisticRegression(penalty='l2', solver='lbfgs'), cv=cv_close_shuffle, param_grid=param_grid, scoring='neg_log_loss')
grid_lr.fit(X_trainingSet, y_trainingSet_quantizing)

# Best C value after rerun the GridSearchCV:
best_C_rerun = grid_lr.best_params_['C']
print(f"\033[1mQ2(d) After Re-run Best C value: {best_C_rerun}\033[0m")
best_log_loss = -grid_lr.best_score_
print(f"\033[1mQ2(d) After Re-run Best log-loss value: {-grid_lr.best_score_}\033[0m")
```

(e)

Suppose that you were going to solve (1) using gradient descent and chose to update each coordinate individually:

- I did this by hand draft. My working is shown as the **three** pictures below:



(e).

From equation (1)

$$L(\beta_0,\beta) = \text{penalty}(\beta) + \frac{\lambda}{n}\sum_{i=1}^{n}\left[y_i \ln\left(\frac{1}{\sigma(\beta_0+\bar{\beta}^T\bar{x}_i)}\right) + (1-y_i)\ln\left(\frac{1}{1-\sigma(\beta_0+\bar{\beta}^T\bar{x}_i)}\right)\right]$$

According to question (a), since we use l_2 regularization, we know (1) is the same as

$$L(\beta_0,\beta) = \frac{1}{2}\|\beta\|_2^2 + \frac{\lambda}{n}\sum_{i=1}^{n}\left[-y_i(\ln\sigma(z_i) - (1-y_i)(\ln(1-\sigma(z_i))\right].$$

which $\frac{1}{2}\|\beta\|_2^2$ is the penalty term and $\frac{\lambda}{n}\sum_{i=1}^{n}\left[-y_i(\ln\sigma(z_i) - (1-y_i)(\ln(1-\sigma(z_i))\right]$ is the loss term.

when referring to β_0, it only exists in the loss term.

For β_0: we know $\beta_0^{(k)} = \beta_0^{(k-1)} - \eta\cdot\dagger$

To calculate the '$\dagger$' part of β_0, we should only look on the loss term

so, by utilizing the chain rule,

we know $\frac{\partial L}{\beta_0} = \frac{\partial L}{\partial z_i}\cdot\frac{\partial z_i}{\partial \beta_0}$.

We should get $\frac{\partial L}{\partial z_i}$ first. We know $z_i = \beta_0 + \bar{\beta}^T\bar{x}_i$.

$$\frac{\partial L}{z_i} = \left(\frac{\lambda}{n}\sum_{i=1}^{n}(-y_i\cdot\ln\sigma(z_i) - (1-y_i)\ln(1-\sigma(z_i)))\right)' = \frac{\lambda}{n}\sum_{i=1}^{n}\left(-y_i\frac{(\sigma(z_i))'}{\sigma(z_i)} - \frac{(1-y_i)}{1-\sigma(z_i)}\cdot(-\sigma(z_i)')\right.$$

Also we know $(\sigma(z_i))' = \sigma(z_i)\cdot(1-\sigma(z_i))$

$$\frac{\partial L}{z_i} = \frac{\lambda}{n}\sum_{i=1}^{n}\left(-y_i\cdot\frac{\sigma(z_i)\cdot(1-\sigma(z_i))}{\sigma(z_i)} - \frac{(1-y_i)}{(1-\sigma(z_i))}\cdot\sigma(z_i)\cdot(1-\sigma(z_i)\right.$$

$$\frac{\partial L}{z_i} = \frac{\lambda}{n}\sum_{i=1}^{n}\left(-y_i\cdot(1-\sigma(z_i)) - (1-y_i)\cdot\sigma(z_i)\right) = \frac{\lambda}{n}\sum_{i=1}^{n}(\sigma(z_i) - y_i)) \cdots (1)$$

And since $z_i = \beta_0 + \bar{\beta}^T\bar{x}_i$ $\therefore \frac{\partial(z_i)}{\partial\beta_0} = 1 \cdots (2)$

Previously by chain rule and ①, ②.

$$\frac{\partial L}{\partial \beta_0} = \frac{\partial L}{\partial z_i} \cdot \frac{\partial z_i}{\partial \beta_0} = \frac{\lambda}{n}\sum_{i=1}^{n}(\sigma(z_i) - y_i) \cdot 1$$

$$\frac{\partial L}{\partial \beta_0} = \frac{\lambda}{n}\sum_{i=1}^{n}(\sigma(z_i) - y_i)$$

so finally we can gee $\beta_0^{(k)} = \beta_0^{(k-1)} - \eta\left(\frac{\lambda}{n}\sum_{i=1}^{n}(\sigma(z_i)^{(k-1)} - y_i)\right)$

For β_j (exclude β_0): $j \in \{1, 2, 3 \cdots p\}$

β_j is different case compared with β_0 because β_j excists in penalty term.

$$L(\beta_0, \beta) = \frac{1}{2}\|\beta\|_2^2 + \frac{\lambda}{n}\sum_{i=1}^{n}[-y_i(\ln\sigma(z_i)) - (1-y_i)(\ln(1-\sigma(z_i)))]$$

similarly to β_0, we first look on the loss term.

we let L_1 = loss term $= \frac{\lambda}{n}\sum_{i=1}^{n}[-y_i(\ln\sigma(z_i)) - (1-y_i)(\ln(1-\sigma(z_i)))]$.

From previous, $\frac{\partial(L_1)}{\partial z_i} = \frac{\lambda}{n}\sum_{i=1}^{n}(\sigma(z_i) - y_i)$

$z_i = \beta_0 + \beta^T x_i$ $\quad \frac{\partial(z_i)}{\partial(\beta_j)} = x_{ij}$ Also by chain rule, we know

$$\frac{\partial(L_1)}{\partial(\beta_j)} = \frac{\partial(L_1)}{\partial(z_i)} \cdot \frac{\partial(z_i)}{\partial(\beta_j)} = \frac{\lambda}{n}\sum_{i=1}^{n}((\sigma(z_i) - y_i) \cdot x_{ij}) \cdots ③$$

For the penalty term: $\frac{1}{2}\|\beta\|_2^2$

$$\frac{\partial(\frac{1}{2}\|\beta\|_2^2)}{\partial\beta_j} = \beta_j \cdots ④$$

By combining ③ ④, we can get

$$\frac{\partial L}{\partial \beta_j} = \left(\frac{\lambda}{n}\sum_{i=1}^{n}[(\sigma(z_i) - y_i) \cdot x_{ij}]\right) + \beta_j \cdots ⑤$$

According to ⑤. we can get.

$$\beta_j^{(k)} = \beta_j^{(k-1)} - \eta\left(\beta_j^{(k-1)} + \frac{\lambda}{n}\sum_{i=1}^{n}\left(\sigma(z_i^{(k-1)}) - y_i\right)\cdot x_{ij}\right)$$

So the final answer is:

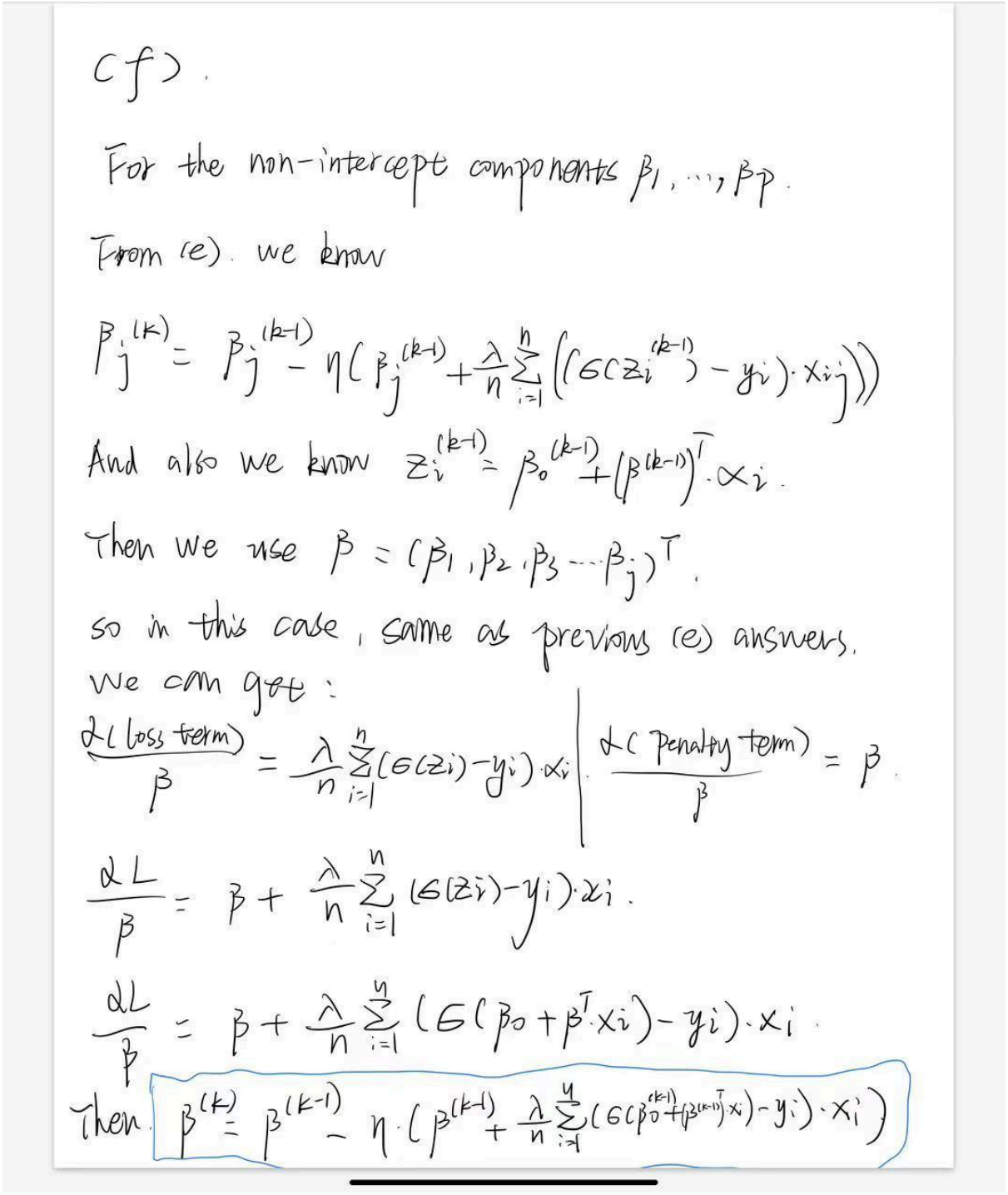
$$\beta_0^{(k)} = \beta_0^{(k-1)} - \eta\left(\frac{\lambda}{n}\sum_{i=1}^{n}\left(\sigma(z_i^{(k-1)}) - y_i\right)\right)$$

$$\beta_j^{(k)} = \beta_j^{(k-1)} - \eta\left(\beta_j^{(k-1)} + \frac{\lambda}{n}\sum_{i=1}^{n}\left(\sigma(z_i^{(k-1)}) - y_i\right)\cdot x_{ij}\right)$$

(f)

Your expression should only be in terms of β0,β,xi and yi. Next, let γ = [β0,βT]T be the (p + 1) dimensional vector that combines the intercept with the coefficient vector β, write down the update:

- I did this by hand draft. My working is shown as the **two** pictures below:



(f).

For the non-intercept components $\beta_1, \dots, \beta_p$.

From (e). we know

$$\beta_j^{(k)} = \beta_j^{(k-1)} - \eta\left(\beta_j^{(k-1)} + \frac{\lambda}{n}\sum_{i=1}^{n}\left(\left(\sigma(z_i^{(k-1)}) - y_i\right)\cdot x_{ij}\right)\right)$$

And also we know $z_i^{(k-1)} = \beta_0^{(k-1)} + \left(\beta^{(k-1)}\right)^T \cdot x_i$.

Then we use $\beta = (\beta_1, \beta_2, \beta_3 \cdots \beta_j)^T$.

so in this case, same as previous (e) answers.

we can get:

$$\frac{\partial L(\text{loss term})}{\beta} = \frac{\lambda}{n}\sum_{i=1}^{n}(\sigma(z_i) - y_i)\cdot x_i \quad \left| \quad \frac{\partial L(\text{penalty term})}{\beta} = \beta \right.$$

$$\frac{\partial L}{\beta} = \beta + \frac{\lambda}{n}\sum_{i=1}^{n}(\sigma(z_i) - y_i)\cdot x_i.$$

$$\frac{\partial L}{\beta} = \beta + \frac{\lambda}{n}\sum_{i=1}^{n}(\sigma(\beta_0 + \beta^T \cdot x_i) - y_i)\cdot x_i.$$

Then $\beta^{(k)} = \beta^{(k-1)} - \eta\cdot\left(\beta^{(k-1)} + \frac{\lambda}{n}\sum_{i=1}^{n}(\sigma(\beta_0^{(k-1)} + (\beta^{(k-1)})^T \cdot x_i) - y_i)\cdot x_i\right)$

The we let $\gamma = [\beta_0, \beta^T]^T = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_j \end{bmatrix}$, $j \in \{1, 2, 3 \cdots p\}$

To match γ We let $x_i' = \begin{bmatrix} 1 \\ x_{i1} \\ x_{i2} \\ \vdots \\ x_{ij} \end{bmatrix}$

· So now we can say $\gamma^T \cdot x_i' = \beta_0 + \beta^T x_i$.

And we can update $z_i = \gamma^T \cdot x_i'$.

Since We should not perform L_2 regularization on β_0, to solve that we plan to use a diagonal matrix Λ to avoid that.

Now the penalty term will be $\frac{1}{2}\gamma^T \cdot \Lambda \cdot \gamma = ||\beta||_2^2$.

Now we can update the original equation to:

$$L(\gamma) = \frac{1}{2}\gamma^T \Lambda \cdot \gamma + \frac{\lambda}{n} \cdot \sum_{i=1}^{n} \left(-y_i \cdot \ln(\sigma(\gamma^T \cdot x_i')) - (1-y_i) \cdot \ln(1-\sigma(\gamma^T \cdot x_i'))\right)$$

$$\gamma^{(k)} = \gamma^{(k-1)} - \eta \cdot \frac{\partial}{\partial}$$

Similar to (e), we can get the new '$\frac{\partial}{\partial}$' value, which equals to $(\Lambda \cdot \gamma^{(k-1)} + \frac{\lambda}{n} \sum_{i=1}^{n} (\sigma((\gamma^{(k-1)})^T \cdot x_i') - y_i) \cdot x_i'))$.

Finally we can get

$$\gamma^{(k)} = \gamma^{(k-1)} - \eta \cdot (\Lambda \cdot \gamma^{(k-1)} + \frac{\lambda}{n} \sum_{i=1}^{n} (\sigma((\gamma^{(k-1)})^T \cdot x_i') - y_i) \cdot x_i')$$