Machine Learning Lab

Simran Kaur

311443

Lab 5

Exercise 1: Backward search for variable selection

Importing required libraries

In [362…

```
import pandas as pd
import numpy as np
```

Reading CSV file

In [363…

```
f = pd.read_csv('bank.csv', sep = ";")
f
```

Out[363…

	age	job	marital	education	default	balance	housing	loan	contact	day	mont
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	o
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	ma
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	a
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	ju
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	ma
...	...	...	...	...	...	...	...	...	...	...	
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	j
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	ma
4518	57	technician	married	secondary	no	295	no	no	cellular	19	au
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	fe
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	a

4521 rows × 17 columns

To convert non-numerical values, I have first taken all possible values occuring in each column and then used a dictionary to give them numerical values.

In [365…

```
f["job"].value_counts()
```

Out[365…

```
management       969
blue-collar      946
technician       768
admin.           478
services         417
retired          230
self-employed    183
entrepreneur     168
unemployed       128
housemaid        112
student           84
unknown           38
Name: job, dtype: int64
```

In [366…

```
f["marital"].value_counts()
```

Out[366…

```
married     2797
single      1196
divorced     528
Name: marital, dtype: int64
```

In [369…

```
f["education"].value_counts()
```

Out[369…

```
secondary    2306
tertiary     1350
primary       678
unknown       187
Name: education, dtype: int64
```

In [370…

```
f["contact"].value_counts()
```

Out[370…

```
cellular     2896
unknown      1324
telephone     301
Name: contact, dtype: int64
```

In [371…

```
f["month"].value_counts()
```

Out[371…

```
may    1398
jul     706
aug     633
jun     531
nov     389
apr     293
feb     222
jan     148
oct      80
sep      52
mar      49
dec      20
Name: month, dtype: int64
```

In [372…

```
f["poutcome"].value_counts()
```

Out[372…

```
unknown    3705
failure     490
other       197
```

```
success      129
Name: poutcome, dtype: int64
```

Giving numerical values to the entries of columns

In [373…

```
values = {"no":0, "yes":1, "management":2, "blue-collar":3, "technician":4, "admin."
         , "entrepreneur":9, "unemployed":10, "housemaid":11, "student":12, "unknown
         ,"secondary":2, "tertiary":3, "primary":4, "cellular":2, "telephone":3, "ja
         , "jul":7, "aug":8, "sep":9, "oct":10, "nov":11, "dec":12, "failure":2, "ot
bank = f.replace(values)
bank
```

Out[373…

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duratio
0	30	10	2	4	0	1787	0	0	2	19	10	7
1	33	6	2	2	0	4789	1	1	2	11	5	22
2	35	2	3	3	0	1350	1	0	2	16	4	18
3	30	2	2	3	0	1476	1	1	13	3	6	19
4	59	3	2	2	0	0	1	0	13	5	5	22
...	...	...	...	...	...	...	...	...	...	...	...	
4516	33	6	2	2	0	-333	1	0	2	30	7	32
4517	57	8	2	3	1	-3313	1	1	13	9	5	15
4518	57	4	2	2	0	295	0	0	2	19	8	15
4519	28	3	2	2	0	1137	0	0	2	6	2	12
4520	44	9	3	3	0	1136	1	1	2	3	4	34

4521 rows × 17 columns

Checking if the rows of the dataframe has any missing or NA values

In [377…

```
is_NA = bank.isna()
row_has_NA = is_NA.any(axis=1)
rows_with_NA = bank[row_has_NA]
print(rows_with_NA)
```

```
Empty DataFrame
Columns: [age, job, marital, education, default, balance, housing, loan, contact, da
y, month, duration, campaign, pdays, previous, poutcome, y]
Index: []
```

There are no such rows

This function is for splitting the dataset into 80% training and 20% test set.

In [378…

```
def split(file):
    r, c = np.shape(file)
    size = int(0.8*r)
    train = file.iloc[0:size]
    test = file.iloc[size :]
    return train, test
```

As we can see some of the columns in our dataframe has very low range values and some have values in a bigger range. This means we need to normalize our dataframe. For this we will first make a copy of our dataframe excluding the last column which we do not want to normalize as it contain classes and normalize rest of the dataframe.

In [379...

```
temp_bank = bank.iloc[:, :-1]
temp_bank
```

Out[379...

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duratic
0	30	10	2	4	0	1787	0	0	2	19	10	
1	33	6	2	2	0	4789	1	1	2	11	5	2
2	35	2	3	3	0	1350	1	0	2	16	4	1
3	30	2	2	3	0	1476	1	1	13	3	6	1
4	59	3	2	2	0	0	1	0	13	5	5	2
...	...	...	...	...	...	...	...	...	...	...	...	
4516	33	6	2	2	0	-333	1	0	2	30	7	3
4517	57	8	2	3	1	-3313	1	1	13	9	5	1
4518	57	4	2	2	0	295	0	0	2	19	8	1
4519	28	3	2	2	0	1137	0	0	2	6	2	1
4520	44	9	3	3	0	1136	1	1	2	3	4	3

4521 rows × 16 columns

In [380...

```
nor_bank =(temp_bank - temp_bank.mean())/temp_bank.std()
```

In [125...

```
nor_bank
```

Out[125...

	age	job	marital	education	default	balance	housing	loan	conta
0	-1.056153	1.972917	-0.716234	0.432252	-0.130744	0.121058	-1.141925	-0.424709	-0.6616
1	-0.772497	0.478228	-0.716234	-0.481144	-0.130744	1.118521	0.875521	2.354032	-0.6616
2	-0.583394	-1.016461	0.721640	-0.024446	-0.130744	-0.024142	0.875521	-0.424709	-0.6616
3	-1.056153	-1.016461	-0.716234	-0.024446	-0.130744	0.017724	0.875521	2.354032	1.5518
4	1.685850	-0.642789	-0.716234	-0.481144	-0.130744	-0.472701	0.875521	-0.424709	1.5518
...	...	...	...	...	...	...	...	...	...
4516	-0.772497	0.478228	-0.716234	-0.481144	-0.130744	-0.583345	0.875521	-0.424709	-0.6616
4517	1.496746	1.225572	-0.716234	-0.024446	7.646823	-1.573497	0.875521	2.354032	1.5518
4518	1.496746	-0.269117	-0.716234	-0.481144	-0.130744	-0.374682	-1.141925	-0.424709	-0.6616
4519	-1.245256	-0.642789	-0.716234	-0.481144	-0.130744	-0.094914	-1.141925	-0.424709	-0.6616
4520	0.267573	1.599245	0.721640	-0.024446	-0.130744	-0.095247	0.875521	2.354032	-0.6616

4521 rows × 16 columns

To this normalized dataframe, we will now add the last column.

In [381…

```
nor_bank["y"] = bank.iloc[:, -1]
nor_bank
```

Out[381…

	age	job	marital	education	default	balance	housing	loan	conta
0	-1.056153	1.972917	-0.716234	0.432252	-0.130744	0.121058	-1.141925	-0.424709	-0.6616
1	-0.772497	0.478228	-0.716234	-0.481144	-0.130744	1.118521	0.875521	2.354032	-0.6616
2	-0.583394	-1.016461	0.721640	-0.024446	-0.130744	-0.024142	0.875521	-0.424709	-0.6616
3	-1.056153	-1.016461	-0.716234	-0.024446	-0.130744	0.017724	0.875521	2.354032	1.5518
4	1.685850	-0.642789	-0.716234	-0.481144	-0.130744	-0.472701	0.875521	-0.424709	1.5518
...	...	...	...	...	...	...	...	...	
4516	-0.772497	0.478228	-0.716234	-0.481144	-0.130744	-0.583345	0.875521	-0.424709	-0.6616
4517	1.496746	1.225572	-0.716234	-0.024446	7.646823	-1.573497	0.875521	2.354032	1.5518
4518	1.496746	-0.269117	-0.716234	-0.481144	-0.130744	-0.374682	-1.141925	-0.424709	-0.6616
4519	-1.245256	-0.642789	-0.716234	-0.481144	-0.130744	-0.094914	-1.141925	-0.424709	-0.6616
4520	0.267573	1.599245	0.721640	-0.024446	-0.130744	-0.095247	0.875521	2.354032	-0.6616

4521 rows × 17 columns

We will convert this dataframe to numpy matrix with a bias column added for the linear combination in the logistic regression part.

In [382…

```
nor_bank_matrix = nor_bank.to_numpy()
bias_col = np.ones(shape=(nor_bank_matrix.shape[0],1))
nor_bank_matrix = np.append(bias_col,nor_bank_matrix,axis=1)
nor_bank_matrix_X = nor_bank_matrix[:, :-1]
nor_bank_matrix_Y = nor_bank_matrix[:, -1]
```

Implementing Logistic Regression with mini batch gradient ascent

In [495…

```
def sigmoid(X, beta):
    sig = (1/(1 + np.exp(-(np.dot(X, beta))))).reshape(-1,1)
    return sig

def log_likelihood(X, Y, beta):
    s = 0
    for i in range(len(X)):
        s += Y[i]*np.dot(X[i], beta) - np.log(1 + np.exp(np.dot(X[i], beta)))
    return s[0]

def log_likelihood_grad(X, Y, beta):
    return np.dot(X.T, np.subtract(Y , sigmoid(X, beta)))

def mini_batch_gradient_ascent(X, Y, i_max, alpha):
    m, n = np.shape(X)
    beta = np.zeros((n,1))
    beta_new = np.zeros((n, 1))
```

```
    batch_size = 50
    l = int(m/batch_size)
    for i in range(l):
        for k in range(i_max):
            beta_new = beta + alpha*(log_likelihood_grad(X[i:(i+1)*batch_size][:], Y
        beta = beta_new
    return beta_new
```

Backward search for variable selection using AIC:

min AIC = −2logL + 2p

We need to minimize AIC, In the backward search we will take each variable and remove one by one from the set of variables and check for the AIC. The rule is that if by removing a variable the AIC decreases then we will remove that variable and this process will continue until the AIC doesnot decrease anymore after removing a variable.

In [391…

```
def AIC(X, Y, beta):
    p = X.shape[1]
    return -2*(log_likelihood(X, Y, beta)) + 2*p

def backward_search(X, Y):
    i_max = 100
    alpha = 10**(-5)
    beta = mini_batch_gradient_ascent(X, Y, i_max, alpha)
    col_used = [i for i in range(X.shape[1])]
    min_ = []
    improvement = True
    begin_AIC = AIC(X, Y, beta)
    while(improvement):
        n = X.shape[1]
        for i in range(n):
            l = [k for k in range(n) if k!= i]
            beta_ = mini_batch_gradient_ascent(X[:, l], Y, i_max, alpha)
            new_AIC = AIC(X[:, l], Y, beta_)
            min_.append(new_AIC)
        if min(min_) < begin_AIC:
            aic, val =  min((v, i) for (i, v) in enumerate(min_))
            begin_AIC = aic
            col_used = np.delete(col_used, val)
            X = np.delete(X, val,1)
            min_ = []
        else:
            improvement = False
    return col_used
```

This function extracts the training and test sets features and target.

In [392…

```
def Extract(f, col_name):
    train, test = split(f)
    r, c = np.shape(train)
    x_train = train.loc[:, train.columns != col_name]
    x_train = x_train.to_numpy()
    bias_column1 = np.ones(shape=(r,1))
    X_train = np.append(bias_column1,x_train,axis=1)
    y_train = train.loc[:, train.columns == col_name]
    Y_train = y_train.to_numpy()
    k, l = np.shape(test)
    x_test = test.loc[:, test.columns != col_name]
```

```
    x_test = x_test.to_numpy()
    bias_column2 = np.ones(shape=(k,1))
    X_test = np.append(bias_column2,x_test,axis=1)
    y_test = test.loc[:, test.columns == col_name]
    Y_test = y_test.to_numpy()
    return X_train, Y_train, X_test, Y_test
```

After running backward search on initial set of variables, we get the variables that gives good AIC score.

In [393…

```
X_train, Y_train, X_test, Y_test = Extract(nor_bank, "y")
cols = backward_search(X_train, Y_train)
```

In [394…

```
cols
```

Out[394…

```
array([ 0,  7,  8,  9, 12, 15, 16])
```

Keeping the same variables in the test set and running mini batch gradient ascent on it we will get predictions for the test set that we can compare with the actual values to find the error.

In [397…

```
train_feature = X_train[:, cols]
b = mini_batch_gradient_ascent(train_feature, Y_train, 500, 10**(-5))
test_feature = X_test[:, cols]
y = sigmoid(test_feature, b)
y_pred = np.zeros((len(y), 1))
for i,ele in enumerate(y):
    if ele >= 0.5:
        y_pred[i] = 1
    else:
        y_pred[i] = 0
```

Error on the Test Set

In [399…

```
def error(y_pred, Y_test):
    N = Y_test.shape[0]
    sum = 0
    for i in range(N):
        if y_pred[i] != Y_test[i]:
            sum += 1
    return (sum/N)*100

print(f'Error :{error(y_pred, Y_test):.2f}%')
```

```
Error :11.71%
```

Exercise 2: Regularization for Logistic Regression

Set containing all possible values for alpha and lambda.

In [400…

```
alpha_set = [0.1, 0.5, 0.01, 0.05, 0.001, 0.005, 0.0001, 0.0005]
lambda_set = [np.exp(-5), np.exp(-10), np.exp(-15), np.exp(-20), np.exp(-25)]
hyperpara_set = np.transpose([np.tile(alpha_set, len(lambda_set)), np.repeat(lambda_
hyperpara_set
```

Out[400…

```
array([[1.00000000e-01, 6.73794700e-03],
       [5.00000000e-01, 6.73794700e-03],
       [1.00000000e-02, 6.73794700e-03],
       [5.00000000e-02, 6.73794700e-03],
       [1.00000000e-03, 6.73794700e-03],
       [5.00000000e-03, 6.73794700e-03],
       [1.00000000e-04, 6.73794700e-03],
       [5.00000000e-04, 6.73794700e-03],
       [1.00000000e-01, 4.53999298e-05],
       [5.00000000e-01, 4.53999298e-05],
       [1.00000000e-02, 4.53999298e-05],
       [5.00000000e-02, 4.53999298e-05],
       [1.00000000e-03, 4.53999298e-05],
       [5.00000000e-03, 4.53999298e-05],
       [1.00000000e-04, 4.53999298e-05],
       [5.00000000e-04, 4.53999298e-05],
       [1.00000000e-01, 3.05902321e-07],
       [5.00000000e-01, 3.05902321e-07],
       [1.00000000e-02, 3.05902321e-07],
       [5.00000000e-02, 3.05902321e-07],
       [1.00000000e-03, 3.05902321e-07],
       [5.00000000e-03, 3.05902321e-07],
       [1.00000000e-04, 3.05902321e-07],
       [5.00000000e-04, 3.05902321e-07],
       [1.00000000e-01, 2.06115362e-09],
       [5.00000000e-01, 2.06115362e-09],
       [1.00000000e-02, 2.06115362e-09],
       [5.00000000e-02, 2.06115362e-09],
       [1.00000000e-03, 2.06115362e-09],
       [5.00000000e-03, 2.06115362e-09],
       [1.00000000e-04, 2.06115362e-09],
       [5.00000000e-04, 2.06115362e-09],
       [1.00000000e-01, 1.38879439e-11],
       [5.00000000e-01, 1.38879439e-11],
       [1.00000000e-02, 1.38879439e-11],
       [5.00000000e-02, 1.38879439e-11],
       [1.00000000e-03, 1.38879439e-11],
       [5.00000000e-03, 1.38879439e-11],
       [1.00000000e-04, 1.38879439e-11],
       [5.00000000e-04, 1.38879439e-11]])
```

Now we will perform Mini Batch Gradient ascent for logistic regression with regularization. The training set for evaluation and the test set for prediction is found using Cross validation where we split the original entire set into K = 5 subparts and first take the first part as test and rest as training set and then second as test and remaining as training set and so on. For each pair of α and λ we will apply gradient ascent to calculate beta and then the respective mean of the risk on the test set for each test set given by cross validation. The pair of α and λ which gives the minimum risk is the one to be chosen.

I have used reg = regularizer for λ

Same loglikelihood function with just regularization term added.

In [494…

```
def log_likelihood_reg(X, Y, beta, reg):
    s = 0
    for i in range(len(X)):
        s += Y[i]*np.dot(X[i], beta) - np.log(1 + np.exp(np.dot(X[i], beta)))
    return (s[0] - reg*(np.sum(beta**2)))

def log_likelihood_reg_grad(X, Y, beta, reg):
    Y = Y.reshape(-1,1)
```

```
    return (np.dot(X.T, np.subtract(Y , sigmoid(X, beta))) - 2*reg*beta)

def mini_batch_gradient_ascent_reg(X, Y, i_max, alpha, reg):
    m, n = np.shape(X)
    beta = np.zeros((n, 1))
    beta_new = np.zeros((n, 1))
    batch_size = 50
    l = int(m/batch_size)
    for k in range(i_max):
        for i in range(l):
            beta_new = beta + alpha*(log_likelihood_reg_grad(X[i:(i+1)*batch_size][:
            beta = beta_new
    return beta_new
```

The function crossval_split returns the kth features and target for both training and test set with K = 5 splits.

The function hyperparameter_tuning performs the gradient ascent for each pair of α and λ keeping track of risk on the test set and returns the best hyperparameter.

In [434…

```
def crossval_split(X, y, K, k):
    A = np.zeros((X.shape[0], X.shape[1] + 1))
    A[:, :-1] = X
    A[:, -1] = y
    size = X.shape[0]//K
    B = [0]*K
    for j in range(K):
        B[j] = A[j*size: (j+1)*size][:]
    X_test = B[k][:, :-1]
    y_test = B[k][:, -1]
    trainx = [x[:, :-1] for i,x in enumerate(B) if i!=k]
    X_train = [item for sublist in trainx for item in sublist]
    trainy =  [x[:, -1] for i,x in enumerate(B) if i!=k]
    y_train = [item for sublist in trainy for item in sublist]
    return np.array(X_train), np.array(y_train), np.array(X_test), np.array(y_test)

def pred(x,b):
    y = sigmoid(x, b)
    y_pred = np.zeros((len(y), 1))
    for i,ele in enumerate(y):
        if ele >= 0.5:
            y_pred[i] = 1
        else:
            y_pred[i] = 0
    return y_pred


def hyperparameter_tuning(X, y):
    K = 5
    i_max = 20
    R_hat = [0]*K
    R_hat_mean = []
    train_acc = []
    test_acc = []
    train_ll = []
    test_ll = []
    for i in range(len(hyperpara_set)):
        alpha, reg = hyperpara_set[i]
        for j in range(K):
            xTrain = crossval_split(X,y,K,j)[0]
            yTrain = crossval_split(X,y,K,j)[1]
            xTest = crossval_split(X,y,K,j)[2]
```

```
            yTest = crossval_split(X,y,K,j)[3]
            beta = mini_batch_gradient_ascent_reg(xTrain, yTrain, i_max, alpha, reg)
            y_new = sigmoid(xTest, beta)
            y_pred = np.zeros((len(y_new), 1))
            for i,ele in enumerate(y_new):
                if ele >= 0.5:
                    y_pred[i] = 1
                else:
                    y_pred[i] = 0
            R_hat[j] = error(y_pred, yTest)/100
        R_hat_mean.append([np.mean(R_hat), alpha, reg])
        R_hat = [0]*K
    mini, best_alpha, best_reg = min(R_hat_mean, key=lambda x: x[0])
    return R_hat_mean, best_alpha, best_reg
```

In [496…

```
R_hat_mean,best_alpha, best_reg = hyperparameter_tuning(nor_bank_matrix_X[:,cols], n
R_hat_mean, best_alpha, best_reg
```

```
C:\Users\simra\AppData\Local\Temp/ipykernel_25512/3655892362.py:2: RuntimeWarning: o
verflow encountered in exp
  sig = (1/(1 + np.exp(-(np.dot(X, beta))))).reshape(-1,1)
```

Out[496…

```
([[0.1679203539823009, 0.1, 0.006737946999085467],
  [0.15530973451327434, 0.5, 0.006737946999085467],
  [0.15353982300884955, 0.01, 0.006737946999085467],
  [0.225, 0.05, 0.006737946999085467],
  [0.10862831858407081, 0.001, 0.006737946999085467],
  [0.11150442477876106, 0.005, 0.006737946999085467],
  [0.1081858407079646, 0.0001, 0.006737946999085467],
  [0.1084070796460177, 0.0005, 0.006737946999085467],
  [0.13783185840707962, 0.1, 4.5399929762484854e-05],
  [0.1274336283185841, 0.5, 4.5399929762484854e-05],
  [0.1608407079646018, 0.01, 4.5399929762484854e-05],
  [0.14314159292035397, 0.05, 4.5399929762484854e-05],
  [0.10862831858407081, 0.001, 4.5399929762484854e-05],
  [0.11172566371681417, 0.005, 4.5399929762484854e-05],
  [0.1081858407079646, 0.0001, 4.5399929762484854e-05],
  [0.1084070796460177, 0.0005, 4.5399929762484854e-05],
  [0.12699115044247786, 0.1, 3.059023205018258e-07],
  [0.19579646017699115, 0.5, 3.059023205018258e-07],
  [0.1243362831858407, 0.01, 3.059023205018258e-07],
  [0.12721238938053098, 0.05, 3.059023205018258e-07],
  [0.10862831858407081, 0.001, 3.059023205018258e-07],
  [0.11172566371681417, 0.005, 3.059023205018258e-07],
  [0.1081858407079646, 0.0001, 3.059023205018258e-07],
  [0.1084070796460177, 0.0005, 3.059023205018258e-07],
  [0.134070796460177, 0.1, 2.061153622438558e-09],
  [0.17610619469026548, 0.5, 2.061153622438558e-09],
  [0.11482300884955751, 0.01, 2.061153622438558e-09],
  [0.13495575221238937, 0.05, 2.061153622438558e-09],
  [0.10862831858407081, 0.001, 2.061153622438558e-09],
  [0.11172566371681417, 0.005, 2.061153622438558e-09],
  [0.1081858407079646, 0.0001, 2.061153622438558e-09],
  [0.1084070796460177, 0.0005, 2.061153622438558e-09],
  [0.1471238938053097, 0.1, 1.3887943864964021e-11],
  [0.1652654867256637, 0.5, 1.3887943864964021e-11],
  [0.12367256637168142, 0.01, 1.3887943864964021e-11],
  [0.13628318584070798, 0.05, 1.3887943864964021e-11],
  [0.10862831858407081, 0.001, 1.3887943864964021e-11],
  [0.11172566371681417, 0.005, 1.3887943864964021e-11],
  [0.1081858407079646, 0.0001, 1.3887943864964021e-11],
  [0.1084070796460177, 0.0005, 1.3887943864964021e-11]],
```

```
0.0001,
0.006737946999085467)
```

In [497…

```
z = []
for a,b,c in R_hat_mean:
    z.append(a)
```

In [498…

```
np.array(z).reshape(8,5)
```

Out[498…

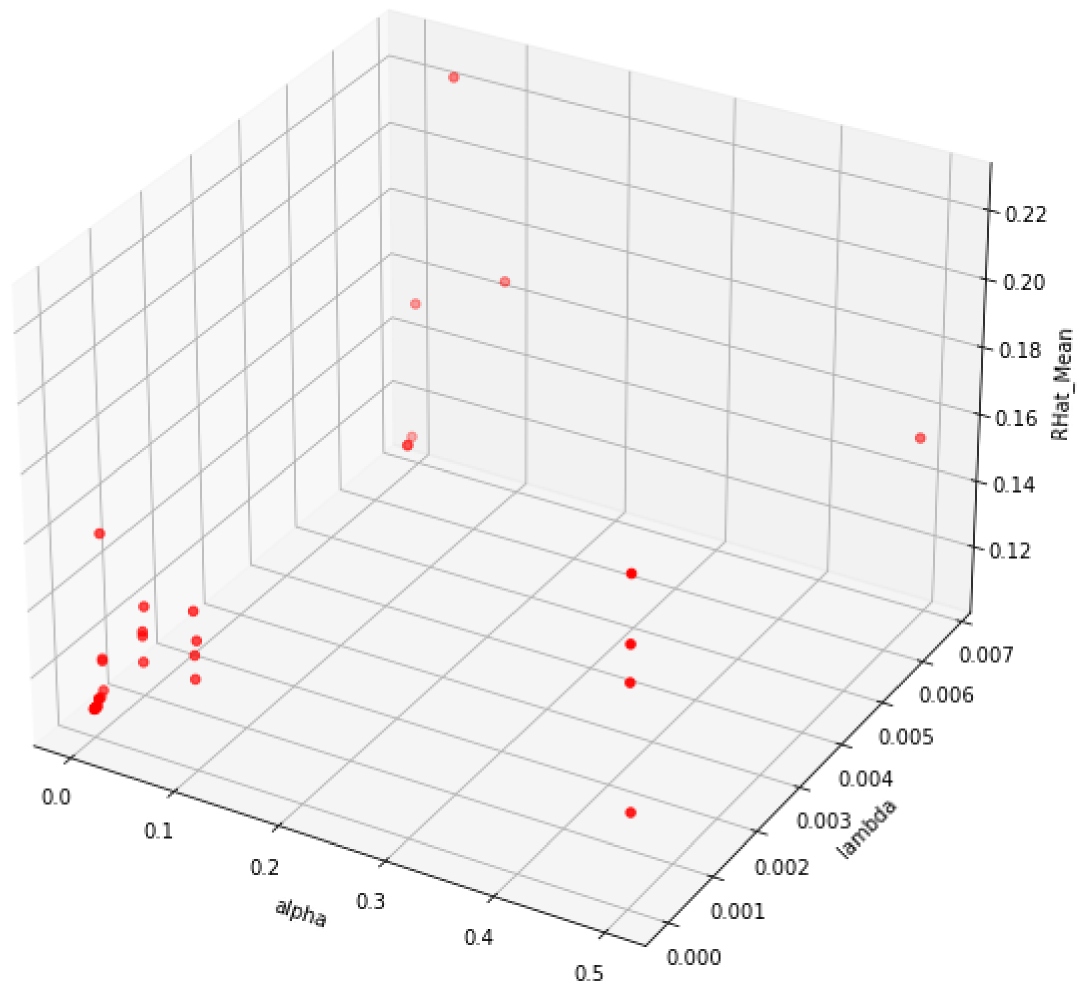
```
array([[0.16792035, 0.15530973, 0.15353982, 0.225     , 0.10862832],
       [0.11150442, 0.10818584, 0.10840708, 0.13783186, 0.12743363],
       [0.16084071, 0.14314159, 0.10862832, 0.11172566, 0.10818584],
       [0.10840708, 0.12699115, 0.19579646, 0.12433628, 0.12721239],
       [0.10862832, 0.11172566, 0.10818584, 0.10840708, 0.1340708 ],
       [0.17610619, 0.11482301, 0.13495575, 0.10862832, 0.11172566],
       [0.10818584, 0.10840708, 0.14712389, 0.16526549, 0.12367257],
       [0.13628319, 0.10862832, 0.11172566, 0.10818584, 0.10840708]])
```

In [499…

```
import matplotlib.pyplot as plt
X, Y = np.meshgrid(alpha_set, lambda_set)
Z = np.array(z).reshape(5,8)
fig = plt.figure(figsize=(10,10))
ax = plt.axes(projection='3d')
# ax.plot_surface(X, Y, Z,alpha=0.5)
ax.scatter3D(X, Y, Z,c="r")
ax.set_xlabel('alpha')
ax.set_ylabel('lambda')
ax.set_zlabel('RHat_Mean')
```

Out[499…

```
Text(0.5, 0, 'RHat_Mean')
```



Finally, for the optimal value of α and λ, we will train our model on complete training data and evaluate on Test data.

In [500…

```
train_features, train_targets, test_features, test_targets = Extract(nor_bank, "y")
```

In [501…

```
beta_grad_ascent = mini_batch_gradient_ascent_reg(train_features, train_targets, 100
beta_grad_ascent
```

Out[501…

```
array([[-2.62545161],
       [ 0.02096983],
       [-0.00519946],
       [ 0.13213052],
       [-0.09823944],
       [ 0.00611378],
       [ 0.06788393],
       [-0.38549186],
       [-0.313453  ],
       [-0.48318793],
       [-0.0787235 ],
       [-0.02648223],
       [ 1.00395384],
       [-0.16358842],
       [-0.34141883],
       [ 0.00551926],
       [-0.62841557]])
```

In [502…
```
test_target_ = sigmoid(test_features, beta_grad_ascent)
test_target_pred = np.zeros((len(test_target_), 1))
for i,ele in enumerate(test_target_):
    if ele >= 0.5:
        test_target_pred[i] = 1
    else:
        test_target_pred[i] = 0
```

Accuracy on the test set

In [503…
```
def accuracy(y_pred, Y_test):
    N = Y_test.shape[0]
    sum = 0
    for i in range(N):
        if y_pred[i] == Y_test[i]:
            sum += 1
    return (sum/N)*100

print(f'Accuracy :{accuracy(test_target_pred, test_targets):.2f}%')
```

```
Accuracy :88.84%
```

Log-likelihood for Test data

In [504…
```
log_likelihood_reg(test_features, test_targets, beta_grad_ascent, best_reg)
```

Out[504…
```
-258.0039952747008
```

Plot Train and Validation Accuracy and Log-likelihood metrics per k − fold iteration.

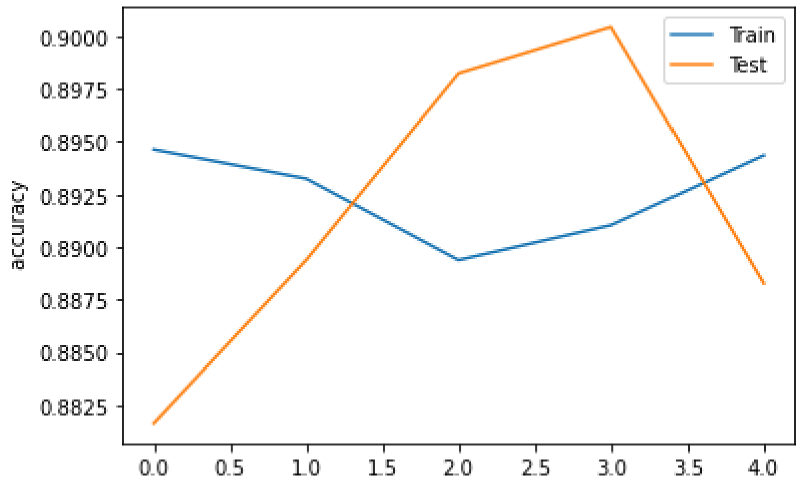
In [505…
```
def K_Fold(X, y, alpha, reg):
    K = 5
    i_max = 20
    train_acc = []
    test_acc = []
    train_ll = []
    test_ll = []
    for j in range(K):
        xTrain = crossval_split(X,y,K,j)[0]
        yTrain = crossval_split(X,y,K,j)[1]
        xTest = crossval_split(X,y,K,j)[2]
        yTest = crossval_split(X,y,K,j)[3]
        beta = mini_batch_gradient_ascent_reg(xTrain, yTrain, i_max, alpha, reg)
        y_pred_train = pred(xTrain,beta)
        y_pred_test = pred(xTest,beta)
        train_acc.append(accuracy(y_pred_train, yTrain)/100)
        test_acc.append(accuracy(y_pred_test, yTest)/100)
        train_ll.append(log_likelihood_reg(xTrain, yTrain, beta, reg))
        test_ll.append(log_likelihood_reg(xTest, yTest, beta, reg))
    return train_acc, test_acc, train_ll, test_ll
```

By taking the best found α and λ in the k Fold iteration the accuracy and loglikelihood for both training and test set are plotted below:

In [506…
```
train_acc, test_acc, train_ll, test_ll = K_Fold(nor_bank_matrix_X, nor_bank_matrix_Y
```

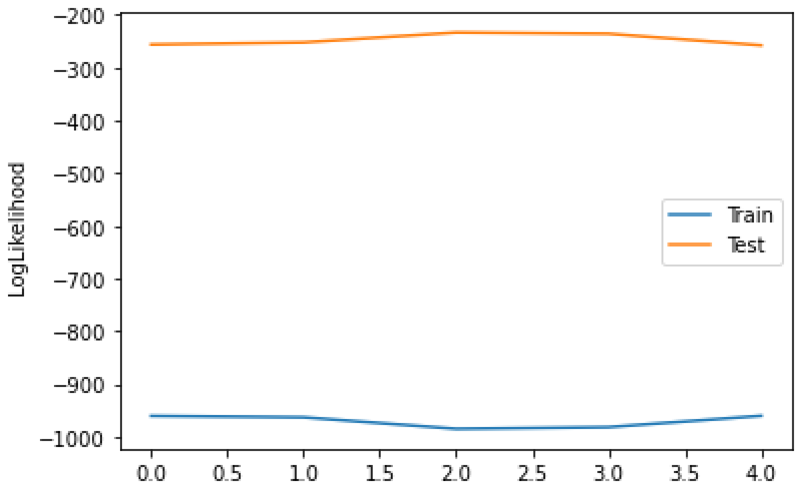
In [507…
```
tr_acc = train_acc
te_acc = test_acc
```

```
plt.plot(tr_acc,label = "Train")
plt.plot(te_acc, label = "Test")
plt.ylabel(' accuracy')
plt.legend()
plt.show()
```



In [508…

```
tr_ll = train_ll
te_ll = test_ll
plt.plot(tr_ll,label = "Train")
plt.plot(te_ll, label = "Test")
plt.ylabel(' LogLikelihood')
plt.legend()
plt.show()
```



Exercise 3: Implementing Hyperband for Logistic Regression

We have taken here possible values of alpha, lambda and batch size.

In [510…

```
alphaSet = [0.1, 0.5, 0.01, 0.05, 0.001, 0.005, 0.0001, 0.0005]
lambdaSet = [np.exp(-5),  np.exp(-8), np.exp(-10), np.exp(-15), np.exp(-18), np.exp(
batchsizeSet = [15, 18, 20, 25, 30, 35, 40, 50]
hyperparaSet = [[a, l, b] for a in alphaSet for l in lambdaSet for b in batchsizeSet
hyperparaSet
```

Out[510…

```
[[0.1, 0.006737946999085467, 15],
 [0.1, 0.006737946999085467, 18],
```

```
[0.1, 0.006737946999085467, 20],
[0.1, 0.006737946999085467, 25],
[0.1, 0.006737946999085467, 30],
[0.1, 0.006737946999085467, 35],
[0.1, 0.006737946999085467, 40],
[0.1, 0.006737946999085467, 45],
[0.1, 0.00033546262790251185, 15],
[0.1, 0.00033546262790251185, 18],
[0.1, 0.00033546262790251185, 20],
[0.1, 0.00033546262790251185, 25],
[0.1, 0.00033546262790251185, 30],
[0.1, 0.00033546262790251185, 35],
[0.1, 0.00033546262790251185, 40],
[0.1, 0.00033546262790251185, 45],
[0.1, 4.5399929762484854e-05, 15],
[0.1, 4.5399929762484854e-05, 18],
[0.1, 4.5399929762484854e-05, 20],
[0.1, 4.5399929762484854e-05, 25],
[0.1, 4.5399929762484854e-05, 30],
[0.1, 4.5399929762484854e-05, 35],
[0.1, 4.5399929762484854e-05, 40],
[0.1, 4.5399929762484854e-05, 45],
[0.1, 3.059023205018258e-07, 15],
[0.1, 3.059023205018258e-07, 18],
[0.1, 3.059023205018258e-07, 20],
[0.1, 3.059023205018258e-07, 25],
[0.1, 3.059023205018258e-07, 30],
[0.1, 3.059023205018258e-07, 35],
[0.1, 3.059023205018258e-07, 40],
[0.1, 3.059023205018258e-07, 45],
[0.1, 1.522997974471263e-08, 15],
[0.1, 1.522997974471263e-08, 18],
[0.1, 1.522997974471263e-08, 20],
[0.1, 1.522997974471263e-08, 25],
[0.1, 1.522997974471263e-08, 30],
[0.1, 1.522997974471263e-08, 35],
[0.1, 1.522997974471263e-08, 40],
[0.1, 1.522997974471263e-08, 45],
[0.1, 2.061153622438558e-09, 15],
[0.1, 2.061153622438558e-09, 18],
[0.1, 2.061153622438558e-09, 20],
[0.1, 2.061153622438558e-09, 25],
[0.1, 2.061153622438558e-09, 30],
[0.1, 2.061153622438558e-09, 35],
[0.1, 2.061153622438558e-09, 40],
[0.1, 2.061153622438558e-09, 45],
[0.1, 2.7894680928689246e-10, 15],
[0.1, 2.7894680928689246e-10, 18],
[0.1, 2.7894680928689246e-10, 20],
[0.1, 2.7894680928689246e-10, 25],
[0.1, 2.7894680928689246e-10, 30],
[0.1, 2.7894680928689246e-10, 35],
[0.1, 2.7894680928689246e-10, 40],
[0.1, 2.7894680928689246e-10, 45],
[0.1, 1.3887943864964021e-11, 15],
[0.1, 1.3887943864964021e-11, 18],
[0.1, 1.3887943864964021e-11, 20],
[0.1, 1.3887943864964021e-11, 25],
[0.1, 1.3887943864964021e-11, 30],
[0.1, 1.3887943864964021e-11, 35],
[0.1, 1.3887943864964021e-11, 40],
[0.1, 1.3887943864964021e-11, 45],
[0.5, 0.006737946999085467, 15],
[0.5, 0.006737946999085467, 18],
```

```
[0.5, 0.006737946999085467, 20],
[0.5, 0.006737946999085467, 25],
[0.5, 0.006737946999085467, 30],
[0.5, 0.006737946999085467, 35],
[0.5, 0.006737946999085467, 40],
[0.5, 0.006737946999085467, 45],
[0.5, 0.0003354626279025118 5, 15],
[0.5, 0.00033546262790251185, 18],
[0.5, 0.00033546262790251185, 20],
[0.5, 0.00033546262790251185, 25],
[0.5, 0.00033546262790251185, 30],
[0.5, 0.00033546262790251185, 35],
[0.5, 0.00033546262790251185, 40],
[0.5, 0.00033546262790251185, 45],
[0.5, 4.5399929762484854e-05, 15],
[0.5, 4.5399929762484854e-05, 18],
[0.5, 4.5399929762484854e-05, 20],
[0.5, 4.5399929762484854e-05, 25],
[0.5, 4.5399929762484854e-05, 30],
[0.5, 4.5399929762484854e-05, 35],
[0.5, 4.5399929762484854e-05, 40],
[0.5, 4.5399929762484854e-05, 45],
[0.5, 3.059023205018258e-07, 15],
[0.5, 3.059023205018258e-07, 18],
[0.5, 3.059023205018258e-07, 20],
[0.5, 3.059023205018258e-07, 25],
[0.5, 3.059023205018258e-07, 30],
[0.5, 3.059023205018258e-07, 35],
[0.5, 3.059023205018258e-07, 40],
[0.5, 3.059023205018258e-07, 45],
[0.5, 1.522997974471263e-08, 15],
[0.5, 1.522997974471263e-08, 18],
[0.5, 1.522997974471263e-08, 20],
[0.5, 1.522997974471263e-08, 25],
[0.5, 1.522997974471263e-08, 30],
[0.5, 1.522997974471263e-08, 35],
[0.5, 1.522997974471263e-08, 40],
[0.5, 1.522997974471263e-08, 45],
[0.5, 2.061153622438558e-09, 15],
[0.5, 2.061153622438558e-09, 18],
[0.5, 2.061153622438558e-09, 20],
[0.5, 2.061153622438558e-09, 25],
[0.5, 2.061153622438558e-09, 30],
[0.5, 2.061153622438558e-09, 35],
[0.5, 2.061153622438558e-09, 40],
[0.5, 2.061153622438558e-09, 45],
[0.5, 2.7894680928689246e-10, 15],
[0.5, 2.7894680928689246e-10, 18],
[0.5, 2.7894680928689246e-10, 20],
[0.5, 2.7894680928689246e-10, 25],
[0.5, 2.7894680928689246e-10, 30],
[0.5, 2.7894680928689246e-10, 35],
[0.5, 2.7894680928689246e-10, 40],
[0.5, 2.7894680928689246e-10, 45],
[0.5, 1.3887943864964021e-11, 15],
[0.5, 1.3887943864964021e-11, 18],
[0.5, 1.3887943864964021e-11, 20],
[0.5, 1.3887943864964021e-11, 25],
[0.5, 1.3887943864964021e-11, 30],
[0.5, 1.3887943864964021e-11, 35],
[0.5, 1.3887943864964021e-11, 40],
[0.5, 1.3887943864964021e-11, 45],
[0.01, 0.006737946999085467, 15],
[0.01, 0.006737946999085467, 18],
```

```
[0.01, 0.006737946999085467, 20],
[0.01, 0.006737946999085467, 25],
[0.01, 0.006737946999085467, 30],
[0.01, 0.006737946999085467, 35],
[0.01, 0.006737946999085467, 40],
[0.01, 0.006737946999085467, 45],
[0.01, 0.00033546262790251185, 15],
[0.01, 0.00033546262790251185, 18],
[0.01, 0.00033546262790251185, 20],
[0.01, 0.00033546262790251185, 25],
[0.01, 0.00033546262790251185, 30],
[0.01, 0.00033546262790251185, 35],
[0.01, 0.00033546262790251185, 40],
[0.01, 0.00033546262790251185, 45],
[0.01, 4.5399929762484854e-05, 15],
[0.01, 4.5399929762484854e-05, 18],
[0.01, 4.5399929762484854e-05, 20],
[0.01, 4.5399929762484854e-05, 25],
[0.01, 4.5399929762484854e-05, 30],
[0.01, 4.5399929762484854e-05, 35],
[0.01, 4.5399929762484854e-05, 40],
[0.01, 4.5399929762484854e-05, 45],
[0.01, 3.059023205018258e-07, 15],
[0.01, 3.059023205018258e-07, 18],
[0.01, 3.059023205018258e-07, 20],
[0.01, 3.059023205018258e-07, 25],
[0.01, 3.059023205018258e-07, 30],
[0.01, 3.059023205018258e-07, 35],
[0.01, 3.059023205018258e-07, 40],
[0.01, 3.059023205018258e-07, 45],
[0.01, 1.522997974471263e-08, 15],
[0.01, 1.522997974471263e-08, 18],
[0.01, 1.522997974471263e-08, 20],
[0.01, 1.522997974471263e-08, 25],
[0.01, 1.522997974471263e-08, 30],
[0.01, 1.522997974471263e-08, 35],
[0.01, 1.522997974471263e-08, 40],
[0.01, 1.522997974471263e-08, 45],
[0.01, 2.061153622438558e-09, 15],
[0.01, 2.061153622438558e-09, 18],
[0.01, 2.061153622438558e-09, 20],
[0.01, 2.061153622438558e-09, 25],
[0.01, 2.061153622438558e-09, 30],
[0.01, 2.061153622438558e-09, 35],
[0.01, 2.061153622438558e-09, 40],
[0.01, 2.061153622438558e-09, 45],
[0.01, 2.7894680928689246e-10, 15],
[0.01, 2.7894680928689246e-10, 18],
[0.01, 2.7894680928689246e-10, 20],
[0.01, 2.7894680928689246e-10, 25],
[0.01, 2.7894680928689246e-10, 30],
[0.01, 2.7894680928689246e-10, 35],
[0.01, 2.7894680928689246e-10, 40],
[0.01, 2.7894680928689246e-10, 45],
[0.01, 1.3887943864964021e-11, 15],
[0.01, 1.3887943864964021e-11, 18],
[0.01, 1.3887943864964021e-11, 20],
[0.01, 1.3887943864964021e-11, 25],
[0.01, 1.3887943864964021e-11, 30],
[0.01, 1.3887943864964021e-11, 35],
[0.01, 1.3887943864964021e-11, 40],
[0.01, 1.3887943864964021e-11, 45],
[0.05, 0.006737946999085467, 15],
[0.05, 0.006737946999085467, 18],
```

```
[0.05, 0.006737946999085467, 20],
[0.05, 0.006737946999085467, 25],
[0.05, 0.006737946999085467, 30],
[0.05, 0.006737946999085467, 35],
[0.05, 0.006737946999085467, 40],
[0.05, 0.006737946999085467, 45],
[0.05, 0.00033546262790251185, 15],
[0.05, 0.00033546262790251185, 18],
[0.05, 0.00033546262790251185, 20],
[0.05, 0.00033546262790251185, 25],
[0.05, 0.00033546262790251185, 30],
[0.05, 0.00033546262790251185, 35],
[0.05, 0.00033546262790251185, 40],
[0.05, 0.00033546262790251185, 45],
[0.05, 4.5399929762484854e-05, 15],
[0.05, 4.5399929762484854e-05, 18],
[0.05, 4.5399929762484854e-05, 20],
[0.05, 4.5399929762484854e-05, 25],
[0.05, 4.5399929762484854e-05, 30],
[0.05, 4.5399929762484854e-05, 35],
[0.05, 4.5399929762484854e-05, 40],
[0.05, 4.5399929762484854e-05, 45],
[0.05, 3.059023205018258e-07, 15],
[0.05, 3.059023205018258e-07, 18],
[0.05, 3.059023205018258e-07, 20],
[0.05, 3.059023205018258e-07, 25],
[0.05, 3.059023205018258e-07, 30],
[0.05, 3.059023205018258e-07, 35],
[0.05, 3.059023205018258e-07, 40],
[0.05, 3.059023205018258e-07, 45],
[0.05, 1.522997974471263e-08, 15],
[0.05, 1.522997974471263e-08, 18],
[0.05, 1.522997974471263e-08, 20],
[0.05, 1.522997974471263e-08, 25],
[0.05, 1.522997974471263e-08, 30],
[0.05, 1.522997974471263e-08, 35],
[0.05, 1.522997974471263e-08, 40],
[0.05, 1.522997974471263e-08, 45],
[0.05, 2.061153622438558e-09, 15],
[0.05, 2.061153622438558e-09, 18],
[0.05, 2.061153622438558e-09, 20],
[0.05, 2.061153622438558e-09, 25],
[0.05, 2.061153622438558e-09, 30],
[0.05, 2.061153622438558e-09, 35],
[0.05, 2.061153622438558e-09, 40],
[0.05, 2.061153622438558e-09, 45],
[0.05, 2.7894680928689246e-10, 15],
[0.05, 2.7894680928689246e-10, 18],
[0.05, 2.7894680928689246e-10, 20],
[0.05, 2.7894680928689246e-10, 25],
[0.05, 2.7894680928689246e-10, 30],
[0.05, 2.7894680928689246e-10, 35],
[0.05, 2.7894680928689246e-10, 40],
[0.05, 2.7894680928689246e-10, 45],
[0.05, 1.3887943864964021e-11, 15],
[0.05, 1.3887943864964021e-11, 18],
[0.05, 1.3887943864964021e-11, 20],
[0.05, 1.3887943864964021e-11, 25],
[0.05, 1.3887943864964021e-11, 30],
[0.05, 1.3887943864964021e-11, 35],
[0.05, 1.3887943864964021e-11, 40],
[0.05, 1.3887943864964021e-11, 45],
[0.001, 0.006737946999085467, 15],
[0.001, 0.006737946999085467, 18],
```

```
[0.001, 0.006737946999085467, 20],
[0.001, 0.006737946999085467, 25],
[0.001, 0.006737946999085467, 30],
[0.001, 0.006737946999085467, 35],
[0.001, 0.006737946999085467, 40],
[0.001, 0.006737946999085467, 45],
[0.001, 0.00033546262790251185, 15],
[0.001, 0.00033546262790251185, 18],
[0.001, 0.00033546262790251185, 20],
[0.001, 0.00033546262790251185, 25],
[0.001, 0.00033546262790251185, 30],
[0.001, 0.00033546262790251185, 35],
[0.001, 0.00033546262790251185, 40],
[0.001, 0.00033546262790251185, 45],
[0.001, 4.5399929762484854e-05, 15],
[0.001, 4.5399929762484854e-05, 18],
[0.001, 4.5399929762484854e-05, 20],
[0.001, 4.5399929762484854e-05, 25],
[0.001, 4.5399929762484854e-05, 30],
[0.001, 4.5399929762484854e-05, 35],
[0.001, 4.5399929762484854e-05, 40],
[0.001, 4.5399929762484854e-05, 45],
[0.001, 3.059023205018258e-07, 15],
[0.001, 3.059023205018258e-07, 18],
[0.001, 3.059023205018258e-07, 20],
[0.001, 3.059023205018258e-07, 25],
[0.001, 3.059023205018258e-07, 30],
[0.001, 3.059023205018258e-07, 35],
[0.001, 3.059023205018258e-07, 40],
[0.001, 3.059023205018258e-07, 45],
[0.001, 1.522997974471263e-08, 15],
[0.001, 1.522997974471263e-08, 18],
[0.001, 1.522997974471263e-08, 20],
[0.001, 1.522997974471263e-08, 25],
[0.001, 1.522997974471263e-08, 30],
[0.001, 1.522997974471263e-08, 35],
[0.001, 1.522997974471263e-08, 40],
[0.001, 1.522997974471263e-08, 45],
[0.001, 2.061153622438558e-09, 15],
[0.001, 2.061153622438558e-09, 18],
[0.001, 2.061153622438558e-09, 20],
[0.001, 2.061153622438558e-09, 25],
[0.001, 2.061153622438558e-09, 30],
[0.001, 2.061153622438558e-09, 35],
[0.001, 2.061153622438558e-09, 40],
[0.001, 2.061153622438558e-09, 45],
[0.001, 2.7894680928689246e-10, 15],
[0.001, 2.7894680928689246e-10, 18],
[0.001, 2.7894680928689246e-10, 20],
[0.001, 2.7894680928689246e-10, 25],
[0.001, 2.7894680928689246e-10, 30],
[0.001, 2.7894680928689246e-10, 35],
[0.001, 2.7894680928689246e-10, 40],
[0.001, 2.7894680928689246e-10, 45],
[0.001, 1.3887943864964021e-11, 15],
[0.001, 1.3887943864964021e-11, 18],
[0.001, 1.3887943864964021e-11, 20],
[0.001, 1.3887943864964021e-11, 25],
[0.001, 1.3887943864964021e-11, 30],
[0.001, 1.3887943864964021e-11, 35],
[0.001, 1.3887943864964021e-11, 40],
[0.001, 1.3887943864964021e-11, 45],
[0.005, 0.006737946999085467, 15],
[0.005, 0.006737946999085467, 18],
```

```
[0.005, 0.006737946999085467, 20],
[0.005, 0.006737946999085467, 25],
[0.005, 0.006737946999085467, 30],
[0.005, 0.006737946999085467, 35],
[0.005, 0.006737946999085467, 40],
[0.005, 0.006737946999085467, 45],
[0.005, 0.00033546262790251185, 15],
[0.005, 0.00033546262790251185, 18],
[0.005, 0.00033546262790251185, 20],
[0.005, 0.00033546262790251185, 25],
[0.005, 0.00033546262790251185, 30],
[0.005, 0.00033546262790251185, 35],
[0.005, 0.00033546262790251185, 40],
[0.005, 0.00033546262790251185, 45],
[0.005, 4.5399929762484854e-05, 15],
[0.005, 4.5399929762484854e-05, 18],
[0.005, 4.5399929762484854e-05, 20],
[0.005, 4.5399929762484854e-05, 25],
[0.005, 4.5399929762484854e-05, 30],
[0.005, 4.5399929762484854e-05, 35],
[0.005, 4.5399929762484854e-05, 40],
[0.005, 4.5399929762484854e-05, 45],
[0.005, 3.059023205018258e-07, 15],
[0.005, 3.059023205018258e-07, 18],
[0.005, 3.059023205018258e-07, 20],
[0.005, 3.059023205018258e-07, 25],
[0.005, 3.059023205018258e-07, 30],
[0.005, 3.059023205018258e-07, 35],
[0.005, 3.059023205018258e-07, 40],
[0.005, 3.059023205018258e-07, 45],
[0.005, 1.522997974471263e-08, 15],
[0.005, 1.522997974471263e-08, 18],
[0.005, 1.522997974471263e-08, 20],
[0.005, 1.522997974471263e-08, 25],
[0.005, 1.522997974471263e-08, 30],
[0.005, 1.522997974471263e-08, 35],
[0.005, 1.522997974471263e-08, 40],
[0.005, 1.522997974471263e-08, 45],
[0.005, 2.061153622438558e-09, 15],
[0.005, 2.061153622438558e-09, 18],
[0.005, 2.061153622438558e-09, 20],
[0.005, 2.061153622438558e-09, 25],
[0.005, 2.061153622438558e-09, 30],
[0.005, 2.061153622438558e-09, 35],
[0.005, 2.061153622438558e-09, 40],
[0.005, 2.061153622438558e-09, 45],
[0.005, 2.7894680928689246e-10, 15],
[0.005, 2.7894680928689246e-10, 18],
[0.005, 2.7894680928689246e-10, 20],
[0.005, 2.7894680928689246e-10, 25],
[0.005, 2.7894680928689246e-10, 30],
[0.005, 2.7894680928689246e-10, 35],
[0.005, 2.7894680928689246e-10, 40],
[0.005, 2.7894680928689246e-10, 45],
[0.005, 1.3887943864964021e-11, 15],
[0.005, 1.3887943864964021e-11, 18],
[0.005, 1.3887943864964021e-11, 20],
[0.005, 1.3887943864964021e-11, 25],
[0.005, 1.3887943864964021e-11, 30],
[0.005, 1.3887943864964021e-11, 35],
[0.005, 1.3887943864964021e-11, 40],
[0.005, 1.3887943864964021e-11, 45],
[0.0001, 0.006737946999085467, 15],
[0.0001, 0.006737946999085467, 18],
```

```
                [0.0001, 0.006737946999085467, 20],
                [0.0001, 0.006737946999085467, 25],
                [0.0001, 0.006737946999085467, 30],
                [0.0001, 0.006737946999085467, 35],
                [0.0001, 0.006737946999085467, 40],
                [0.0001, 0.006737946999085467, 45],
                [0.0001, 0.00033546262790251185, 15],
                [0.0001, 0.00033546262790251185, 18],
                [0.0001, 0.00033546262790251185, 20],
                [0.0001, 0.00033546262790251185, 25],
                [0.0001, 0.00033546262790251185, 30],
                [0.0001, 0.00033546262790251185, 35],
                [0.0001, 0.00033546262790251185, 40],
                [0.0001, 0.00033546262790251185, 45],
                [0.0001, 4.5399929762484854e-05, 15],
                [0.0001, 4.5399929762484854e-05, 18],
                [0.0001, 4.5399929762484854e-05, 20],
                [0.0001, 4.5399929762484854e-05, 25],
                [0.0001, 4.5399929762484854e-05, 30],
                [0.0001, 4.5399929762484854e-05, 35],
                [0.0001, 4.5399929762484854e-05, 40],
                [0.0001, 4.5399929762484854e-05, 45],
                [0.0001, 3.059023205018258e-07, 15],
                [0.0001, 3.059023205018258e-07, 18],
                [0.0001, 3.059023205018258e-07, 20],
                [0.0001, 3.059023205018258e-07, 25],
                [0.0001, 3.059023205018258e-07, 30],
                [0.0001, 3.059023205018258e-07, 35],
                [0.0001, 3.059023205018258e-07, 40],
                [0.0001, 3.059023205018258e-07, 45],
                [0.0001, 1.522997974471263e-08, 15],
                [0.0001, 1.522997974471263e-08, 18],
                [0.0001, 1.522997974471263e-08, 20],
                [0.0001, 1.522997974471263e-08, 25],
                [0.0001, 1.522997974471263e-08, 30],
                [0.0001, 1.522997974471263e-08, 35],
                [0.0001, 1.522997974471263e-08, 40],
                [0.0001, 1.522997974471263e-08, 45],
                [0.0001, 2.061153622438558e-09, 15],
                [0.0001, 2.061153622438558e-09, 18],
                [0.0001, 2.061153622438558e-09, 20],
                [0.0001, 2.061153622438558e-09, 25],
                [0.0001, 2.061153622438558e-09, 30],
                [0.0001, 2.061153622438558e-09, 35],
                [0.0001, 2.061153622438558e-09, 40],
                [0.0001, 2.061153622438558e-09, 45],
                [0.0001, 2.7894680928689246e-10, 15],
                [0.0001, 2.7894680928689246e-10, 18],
                [0.0001, 2.7894680928689246e-10, 20],
                [0.0001, 2.7894680928689246e-10, 25],
                [0.0001, 2.7894680928689246e-10, 30],
                [0.0001, 2.7894680928689246e-10, 35],
                [0.0001, 2.7894680928689246e-10, 40],
                [0.0001, 2.7894680928689246e-10, 45],
                [0.0001, 1.3887943864964021e-11, 15],
                [0.0001, 1.3887943864964021e-11, 18],
                [0.0001, 1.3887943864964021e-11, 20],
                [0.0001, 1.3887943864964021e-11, 25],
                [0.0001, 1.3887943864964021e-11, 30],
                [0.0001, 1.3887943864964021e-11, 35],
                [0.0001, 1.3887943864964021e-11, 40],
                [0.0001, 1.3887943864964021e-11, 45],
                [0.0005, 0.006737946999085467, 15],
                [0.0005, 0.006737946999085467, 18],
```

```
[0.0005, 0.006737946999085467, 20],
[0.0005, 0.006737946999085467, 25],
[0.0005, 0.006737946999085467, 30],
[0.0005, 0.006737946999085467, 35],
[0.0005, 0.006737946999085467, 40],
[0.0005, 0.006737946999085467, 45],
[0.0005, 0.00033546262790251185, 15],
[0.0005, 0.00033546262790251185, 18],
[0.0005, 0.00033546262790251185, 20],
[0.0005, 0.00033546262790251185, 25],
[0.0005, 0.00033546262790251185, 30],
[0.0005, 0.00033546262790251185, 35],
[0.0005, 0.00033546262790251185, 40],
[0.0005, 0.00033546262790251185, 45],
[0.0005, 4.5399929762484854e-05, 15],
[0.0005, 4.5399929762484854e-05, 18],
[0.0005, 4.5399929762484854e-05, 20],
[0.0005, 4.5399929762484854e-05, 25],
[0.0005, 4.5399929762484854e-05, 30],
[0.0005, 4.5399929762484854e-05, 35],
[0.0005, 4.5399929762484854e-05, 40],
[0.0005, 4.5399929762484854e-05, 45],
[0.0005, 3.059023205018258e-07, 15],
[0.0005, 3.059023205018258e-07, 18],
[0.0005, 3.059023205018258e-07, 20],
[0.0005, 3.059023205018258e-07, 25],
[0.0005, 3.059023205018258e-07, 30],
[0.0005, 3.059023205018258e-07, 35],
[0.0005, 3.059023205018258e-07, 40],
[0.0005, 3.059023205018258e-07, 45],
[0.0005, 1.522997974471263e-08, 15],
[0.0005, 1.522997974471263e-08, 18],
[0.0005, 1.522997974471263e-08, 20],
[0.0005, 1.522997974471263e-08, 25],
[0.0005, 1.522997974471263e-08, 30],
[0.0005, 1.522997974471263e-08, 35],
[0.0005, 1.522997974471263e-08, 40],
[0.0005, 1.522997974471263e-08, 45],
[0.0005, 2.061153622438558e-09, 15],
[0.0005, 2.061153622438558e-09, 18],
[0.0005, 2.061153622438558e-09, 20],
[0.0005, 2.061153622438558e-09, 25],
[0.0005, 2.061153622438558e-09, 30],
[0.0005, 2.061153622438558e-09, 35],
[0.0005, 2.061153622438558e-09, 40],
[0.0005, 2.061153622438558e-09, 45],
[0.0005, 2.7894680928689246e-10, 15],
[0.0005, 2.7894680928689246e-10, 18],
[0.0005, 2.7894680928689246e-10, 20],
[0.0005, 2.7894680928689246e-10, 25],
[0.0005, 2.7894680928689246e-10, 30],
[0.0005, 2.7894680928689246e-10, 35],
[0.0005, 2.7894680928689246e-10, 40],
[0.0005, 2.7894680928689246e-10, 45],
[0.0005, 1.3887943864964021e-11, 15],
[0.0005, 1.3887943864964021e-11, 18],
[0.0005, 1.3887943864964021e-11, 20],
[0.0005, 1.3887943864964021e-11, 25],
[0.0005, 1.3887943864964021e-11, 30],
[0.0005, 1.3887943864964021e-11, 35],
[0.0005, 1.3887943864964021e-11, 40],
[0.0005, 1.3887943864964021e-11, 45]]
```

In [486…

```
def resplit(file):
    r, c = np.shape(file)
    size = int(0.7*r)
    train = file.iloc[0:size]
    val = file.iloc[size: size +int( 0.15*r)]
    test = file.iloc[size + int(0.15*r) :]
    return train, val, test

def extract_data(f):
    tr, va, te = resplit(f)
    x_tr = tr.loc[:, tr.columns != "y"]
    x_tr = x_tr.to_numpy()
    bias_col1 = np.ones(shape=(tr.shape[0],1))
    X_tr = np.append(bias_col1,x_tr,axis=1)
    y_tr = tr.loc[:, tr.columns == "y"]
    Y_tr = y_tr.to_numpy()

    x_va = va.loc[:, va.columns != "y"]
    x_va = x_va.to_numpy()
    bias_col2 = np.ones(shape=(va.shape[0],1))
    X_va = np.append(bias_col2, x_va, axis=1)
    y_va = va.loc[:, va.columns == "y"]
    Y_va = y_va.to_numpy()

    x_te = te.loc[:, te.columns != "y"]
    x_te = x_te.to_numpy()
    bias_col3 = np.ones(shape=(te.shape[0],1))
    X_te = np.append(bias_col3, x_te ,axis=1)
    y_te = te.loc[:, te.columns == "y"]
    Y_te = y_te.to_numpy()
    return X_tr, Y_tr, X_va, Y_va, X_te, Y_te
```

In [527…

```
def mini_batch_gradient_ascent_reg_para(X, Y, i_max, alpha, reg, batch_size):
    m, n = np.shape(X)
    beta = np.zeros((n, 1))
    beta_new = np.zeros((n, 1))
    l = int(m/batch_size)
    for k in range(i_max):
        for i in range(l):
            beta_new = beta + alpha*(log_likelihood_reg_grad(X[i*batch_size:(i+1)*ba
            beta = beta_new
    return beta_new
```

Hyperband optimization

In [534…

```
import math
def hyperband(X, Y):
    max_iter = 81
    eta = 3
    logeta = lambda x: np.log(x)/np.log(eta)
    s_max = int(logeta(max_iter))
    B = (s_max+1)*max_iter

    for s in reversed(range(s_max+1)):
        n = int(math.ceil(int(B/max_iter/(s+1))*eta**s))
        r = max_iter*eta**(-s)
        T = [hyperparaSet[i] for i in range(len(hyperparaSet))]
        for i in range(s+1):
            n_i = int(n*eta**(-i))
            r_i = int(r*eta**(i))
            val_losses = [log_likelihood_reg(X, Y, mini_batch_gradient_ascent_reg_pa
```

```
                    for alpha, reg, batch_size in T ]
            #T = [T[:,-1].argsort()]

    return val_losses
```

In [535…

```
X_tr, Y_tr, X_va, Y_va, X_te, Y_te = extract_data(nor_bank)
val = hyperband(X_tr, Y_tr)
val
```

Out[535…

```
[-989.6624261318302,
 -975.7845877653674,
 -1100.119574775414,
 -972.090413012198,
 -1000.7326636698785,
 -1002.7706985022814,
 -1035.94845699472,
 -1038.6948080258483,
 -996.3486092303933,
 -981.1987417162424,
 -1106.9810879160264,
 -976.1434775220596,
 -1003.8359230610868,
 -1005.3664294498714,
 -1040.0049214725402,
 -1040.5471248873355,
 -996.6867957488843,
 -981.4695207507955,
 -1107.3170966748733,
 -976.3416194213024,
 -1003.983065044594,
 -1005.4888164240697,
 -1040.1973307363996,
 -1040.635599218635,
 -996.739683422967,
 -981.5118408947284,
 -1107.3695472041002,
 -976.3725465230858,
 -1004.0059969457626,
 -1005.5078832079869,
 -1040.227313441063,
 -1040.6493925793097,
 -996.740024606887,
 -981.5121138836731,
 -1107.36988548314,
 -976.3727459847842,
 -1004.006144812718,
 -1005.5080061460602,
 -1040.2275067684704,
 -1040.6494815242568,
 -996.7400400642075,
 -981.512126251418,
 -1107.369900808832,
 -976.3727550213657,
 -1004.0061515118009,
 -1005.5080117157448,
 -1040.227515527133,
 -1040.6494855538922,
 -996.7400421561289,
 -981.5121279252065,
 -1107.3699028829396,
 -976.3727562443387,
 -1004.0061524184254,
```

```
-1005.508012469518,
-1040.227516712485,
-1040.6494860992436,
-996.7400424672525,
-981.5121281741443,
-1107.3699031914123,
-976.3727564262249,
-1004.0061525532619,
-1005.5080125816237,
-1040.2275168887795,
-1040.6494861803506,
-2801.5971109635084,
-2017.7618574239,
-3360.9407275327717,
-2503.6431927531353,
-3179.2228792060387,
-2687.5629129656418,
-2532.8286308282836,
-2595.109482749702,
-2599.447836071409,
-2254.7972061974033,
-3493.1145619571466,
-2540.1042432353343,
-3253.3415563560775,
-2675.137988852365,
-2422.781760511855,
-3004.068500362594,
-2748.6112649958454,
-2273.192785353849,
-3531.331699139801,
-2736.9035598811824,
-3135.4915382654694,
-2656.4004271474473,
-2782.9877266803287,
-3026.166555151669,
-2758.011416137086,
-2278.5680098852185,
-3555.1149731520027,
-2619.1289048329013,
-3256.4399459696047,
-2683.3188859806783,
-2727.7759398151857,
-4022.722320149906,
-2647.947188154888,
-2278.604261819403,
-3555.231870406429,
-2809.2771804342324,
-2850.276228857647,
-2683.33985874689,
-2591.3385369449293,
-3025.056857064298,
-2708.0904284611333,
-2278.605904612301,
-3555.2371583268728,
-2686.1921886058326,
-2864.5795618572483,
-2683.3408094544257,
-2720.7267318513314,
-3141.4053172953527,
-2689.9888900125593,
-2278.6061269428546,
-3555.2378739153446,
-2742.9416118837717,
-2765.4138565111366,
```

```
-2683.340938122306,
-3204.573426926333,
-3948.037687798332,
-2904.49780698589,
-2278.6061600090475,
-3555.2379803402087,
-2746.2418226034924,
-3368.1462063268186,
-2683.3409572585238,
-2534.994958439292,
-3754.1574850901625,
-861.6052441970286,
-861.8858272874627,
-862.155061065775,
-862.0327507264012,
-861.8427768730606,
-861.5476883506021,
-862.1536459912943,
-861.6450578291597,
-861.8162021803286,
-862.1161965076108,
-862.2390831828859,
-862.2206166637959,
-862.0292392204319,
-861.6949100067598,
-862.2335431669432,
-861.7808479167668,
-861.8426412673367,
-862.1386520325408,
-862.2520430949146,
-862.23544499017,
-862.0421838957573,
-861.7048587060401,
-862.2395621246027,
-861.789042717354,
-861.8468996508384,
-862.1422464975873,
-862.2541358908551,
-862.2378031673483,
-862.0442335517988,
-861.706432280124,
-862.2405174172907,
-861.7903332882571,
-861.846927230096,
-862.1422697579927,
-862.2541494492565,
-862.2378184143299,
-862.0442467963159,
-861.7064424468114,
-862.240523592144,
-861.7903416216744,
-861.8469284796074,
-862.1422708118259,
-862.2541500635359,
-862.2378191051027,
-862.0442473963651,
-861.7064429074161,
-862.2405238718999,
-861.7903419992226,
-861.8469286487091,
-862.1422709544478,
-862.2541501466687,
-862.2378191985877,
-862.0442474775755,
```

```
-861.7064429697534,
-862.2405239097619,
-861.7903420503185,
-861.8469286738588,
-862.1422709756582,
-862.2541501590352,
-862.2378192124917,
-862.0442474896486,
-861.7064429790234,
-862.2405239153952,
-861.7903420579186,
-883.6342669125034,
-884.6944516524278,
-917.8856721939676,
-883.8690770515943,
-885.5168286693624,
-884.4379532071596,
-899.8630069429266,
-892.5858234436638,
-884.9171745835931,
-885.9841917307839,
-919.1426840738856,
-884.8516619019936,
-886.4461767025789,
-885.0865379310648,
-900.6170156732933,
-893.1063237789004,
-885.0014090462341,
-886.0619271983073,
-919.2144288175385,
-884.906174863232,
-886.4958641304796,
-885.1211377954809,
-900.6553178400465,
-893.1333618053519,
-885.0147353661332,
-886.074179666123,
-919.2257087283732,
-884.9147337058826,
-886.50365189701,
-885.1265597485817,
-900.661306199795,
-893.137593462049,
-885.0148214694609,
-886.0742587915279,
-919.2257815484708,
-884.9147889491644,
-886.5037021516696,
-885.1265947356375,
-900.6613448298257,
-893.1376207636669,
-885.0148253704131,
-886.0742623763266,
-919.2257848475986,
-884.9147914519691,
-886.503704428459,
-885.1265963207277,
-900.6613465799583,
-893.1376220005658,
-885.0148258983464,
-886.0742628614763,
-919.2257852940838,
-884.9147917906859,
-886.5037047365897,
```

```
-885.1265965352468,
-900.6613468168131,
-893.1376221679627,
-885.0148259768624,
-886.0742629336294,
-919.2257853604898,
-884.9147918410613,
-886.5037047824137,
-885.1265965671503,
-900.66134685204,
-893.1376221928613,
-858.3681557655211,
-858.2854745526121,
-858.2221954970905,
-858.1965421923378,
-858.163537807361,
-858.1498966463643,
-858.0812586843888,
-858.1328990007601,
-858.0609394239572,
-858.0631166960392,
-857.9702985480297,
-858.0637356350221,
-858.0611960983582,
-858.0634154373449,
-857.9717829760857,
-858.0649441933759,
-858.061097135294,
-858.0629846162528,
-857.9666578796101,
-858.0629810299063,
-858.0602473125126,
-858.0622178926454,
-857.9688244442564,
-858.0635205350667,
-858.0612437881014,
-858.0630488647745,
-857.9661575603328,
-858.0629074973858,
-858.0601301650913,
-858.0620539378701,
-857.9683807483095,
-858.0633125860066,
-858.0612448406232,
-858.0630493532683,
-857.9661543928805,
-858.0629070617439,
-858.0601294365397,
-858.0620529004716,
-857.9683779024936,
-858.063311257281,
-858.0612448883408,
-858.0630493754194,
-857.966154249396,
-858.0629070420212,
-858.0601294035415,
-858.0620528534791,
-857.9683777735696,
-858.063311197088,
-858.0612448947986,
-858.0630493784204,
-857.9661542299804,
-858.0629070393502,
-858.0601293990748,
```

```
-858.0620528471194,
-857.9683777561213,
-858.0633111889418,
-858.0612448957606,
-858.0630493788651,
-857.9661542270912,
-858.062907038954,
-858.0601293984072,
-858.0620528461735,
-857.9683777535269,
-858.0633111877318,
-860.9145934188691,
-860.9590098008065,
-860.6235184373917,
-860.9633923145858,
-860.8837197438054,
-860.8093941564049,
-860.6287080750919,
-860.8680042414824,
-860.8391205299878,
-860.9378250163319,
-860.5285628342922,
-860.9731631945501,
-860.9097059489908,
-860.8295089954545,
-860.6063416952793,
-860.8917621879339,
-860.8512894632692,
-860.9479210574289,
-860.5328238921785,
-860.9794439386601,
-860.915006377889,
-860.8334510449982,
-860.607554144016,
-860.8947016363174,
-860.8533180386003,
-860.9495858008297,
-860.5335593046642,
-860.9804692783337,
-860.9158645418261,
-860.8340887712769,
-860.6077607870332,
-860.8951737345755,
-860.8533312347504,
-860.9495966151426,
-860.5335641090724,
-860.9804759304457,
-860.9158701033743,
-860.834092903784,
-860.6077621349187,
-860.8951767909066,
-860.8533318326361,
-860.9495971051074,
-860.5335643267565,
-860.9804762318282,
-860.915870355348,
-860.8340930910126,
-860.6077621959885,
-860.8951769293776,
-860.8533319135516,
-860.9495971714167,
-860.5335643562127,
-860.9804762726204,
-860.9158703894503,
```

```
-860.834093116353,
-860.6077622042549,
-860.8951769481173,
-860.8533319255865,
-860.9495971812804,
-860.5335643605977,
-860.9804762786852,
-860.9158703945234,
-860.8340931201232,
-860.6077622054844,
-860.8951769509052,
-881.3048696481654,
-881.0471921713633,
-881.044487377542,
-880.6891365062337,
-880.5342941225795,
-880.4258152499888,
-880.4682204345369,
-880.2776891274627,
-879.8228073600208,
-879.8094341224819,
-879.9300123740858,
-879.7912010845965,
-879.7812144532904,
-879.775895323834,
-879.8963360443425,
-879.7649531845575,
-879.7566725902923,
-879.7540571480889,
-879.8800894973697,
-879.7508796981955,
-879.7473440519825,
-879.7466313686152,
-879.8705647294622,
-879.7418311382154,
-879.7463990652432,
-879.7454535928889,
-879.8723328118156,
-879.7446140375262,
-879.7420804002069,
-879.7420833201157,
-879.8665593215104,
-879.7382373546308,
-879.7463328499343,
-879.745398139907,
-879.8722828167745,
-879.7445736520656,
-879.7420464728422,
-879.7420540050083,
-879.8665335039319,
-879.7382141901361,
-879.7463298500685,
-879.7453956276282,
-879.8722805517666,
-879.7445718224154,
-879.7420449357734,
-879.742052676898,
-879.8665323342744,
-879.7382131406774,
-879.7463294440822,
-879.7453952876317,
-879.8722802452301,
-879.7445715747987,
-879.7420447277557,
```

```
-879.7420524971591,
-879.8665321759792,
-879.7382129986463,
-879.7463293837014,
-879.7453952370605,
-879.872280199641,
-879.7445715379698,
-879.7420446968154,
-879.7420524704248,
-879.8665321524344,
-879.7382129775254,
-858.5797079977261,
-858.4749070037899,
-858.4508804609907,
-858.3518169865679,
-858.3036389282762,
-858.2760747326258,
-858.2523197038854,
-858.2387961050987,
-858.1061275894169,
-858.1038608859772,
-858.0791470655131,
-858.1027648991003,
-858.0997515252114,
-858.1008449606851,
-858.0729728480061,
-858.0986858041547,
-858.0952362749779,
-858.094468489776,
-858.0681690244494,
-858.095378234031,
-858.093232818432,
-858.0949069770913,
-858.0663357744221,
-858.093549091463,
-858.0936318273951,
-858.0930698667173,
-858.0665107002314,
-858.0942616495792,
-858.0922413960244,
-858.0939999415314,
-858.0653158836113,
-858.0927601971913,
-858.0936215630289,
-858.0930609052075,
-858.0665000531409,
-858.0942544799664,
-858.0922350246341,
-858.093994108936,
-858.0653093199052,
-858.0927551205033,
-858.0936210980259,
-858.0930604992241,
-858.0664995707898,
-858.0942541551574,
-858.0922347359868,
-858.0939938446954,
-858.0653090225437,
-858.0927548905081,
-858.0936210350959,
-858.0930604442796,
-858.0664995055121,
-858.0942541111993,
-858.0922346969239,
```

```
 -858.0939938089357,
 -858.0653089823002,
 -858.0927548593818,
 -858.0936210257356,
 -858.0930604361077,
 -858.0664994958012,
 -858.0942541046588,
 -858.092234691115,
 -858.0939938036162,
 -858.0653089763154,
 -858.0927548547529]
```

Best Found hyperparameters are:

In [539…

```
T[np.argmin(val)]
```

Out[539…

```
[0.0001, 4.5399929762484854e-05, 50]
```

In [542…

```
beta_hyper = mini_batch_gradient_ascent_reg_para(X_tr, Y_tr, 100, 0.0001, 4.53999297
te_target_ = sigmoid(X_te, beta_hyper)
te_target_pred = np.zeros((len(te_target_), 1))
for i,ele in enumerate(te_target_):
    if ele >= 0.5:
        te_target_pred[i] = 1
    else:
        te_target_pred[i] = 0
```

In [543…

```
print(f'Accuracy :{accuracy(te_target_pred, Y_te):.2f}%')
```

```
Accuracy :88.66%
```