Student ID: R05944004

Name: Cheng-Wei Wu

hw 2
Spam Classification

Report

• Program Overview

Different from the hw1, I modularized several parts of my program this time. All the functions are show as follows:

1. sigmoid
2. cross_entropy
3. read_data
4. logistic_regression
5. gen_model
6. read_model
7. read_test
8. gen_ans

• sigmoid (Fig. 4)

It's a normal sigmoid function (support numpy array operation).

• cross_entropy (Fig. 5)

It will compute the cross-entropy for each weight. Since the predicted value may very close to either 0 or 1, which causing operation error to the numpy log function, I add a small value (1e-3) to the predicted value which is close to either 0 or 1 to avoid such runtime error. (Fig 2)

• read_data

It's a normal function to read the training data into the format of numpy array. (the data format is same with the hw1 → row: observation, col: variables)

• logistic_regression (Fig. 6)

It's similar with the linear regression version. The only differences are list below:

1. It replaces the results of inner product of weights and data with the same results but under the sigmoid function manipulation.
2. It changes the coefficients of computed gradient value of both weights and bias
3. It changes the way to calculate loss function (root-mean-square error to cross-entropy)

Beyond these difference, I thought that the logistic one is almost the same with linear one.

- **gen_model**

  I used this function to write back the weights and bias to the model file via json format.

- **read_model**

  It's a basic and trivial function which load the generated model.

- **read_test**

  It's almost the same with the function "read_data" which read the testing data.

- **gen_ans (Fig. 7)**

  Like previous functions, this function is similar with the part of the codes in hw1. It calculates the predicted value of testing data and fills the corresponding answer to the array according to their probability.

  Finally, I ran the different functions according to the length of the system parameters and generated the "prediction.csv".

• Learning Rate

Different from the hw1, in this homework, only the adagrad and adamgrad can converge in specific epochs. The results of using adagrad is similar with using adamgrad.

• Discussion

In this assignment, I have implemented 3 different methods:

1. Master → Logistic Regression (#iter: 8000, Adagrad, learning rate: 1 → Accuracy: **0.94333**)
2. Method2 → Multi-Logistic ("Neural Network", 4 layers, 4 neurons → Accuracy: **0.94667**) Fig. 8
3. Method3 → Probabilistic Generative Model (Accuracy: **0.94333**) → Fig. 9
4. Method4 → Linear Regression (#iter: 8000, Adagrad, learning rate: 1 → Accuracy: **0.93667**)

I had tried some different combinations of the parameters such like #iteration, learning rate, feature extraction, etc. The best one's accuracy in the public leaderboard on the Kaggle is 0.95000, but I forgot the setting of the parameters… The other three methods I implemented are multi-logistic, probabilistic generative and linear regression. Take multi-logistic for further discussion, I setup 2 parameters "layer" and "neuron" to determine the structure of my neuron network. Then I pass the data to first layer, and take the output to be the next layer's input and so on. In the last layer, I output the final result to judge whether the mail is spam or not. The whole process is something like continuously doing the simple logistic regression, the only difference is to use the previous layer's output to be the training data for the next layer. Also, the data format should be check carefully; otherwise, the program will crash. For example, the format of the output model should be designed carefully such that it can be easily loaded later (I used nested dictionary). What's more, the keys of the loaded dictionary are all encoded in "unicode", so it need the preprocessing. Additionally, I did some experiments which are show in several line charts on the next page. After doing these experiments, I was attracted by the different feature extraction methods' effect; in other words, the difference between feature scale and combinations could result in different accuracy. Since this difference is bigger than I expected, I decided to try other possible combinations and plotted in the line chart (I only try 3 different methods at first). The reason why I came up with this idea is that I noticed that the last several features represent the rare characters which may not appear in the normal email content. (Note that all the experiments are under normal logistic version) And since I tried to square the values of features and got overflow error, so I tried the root values and log values of features to get the best combination.

- Figures

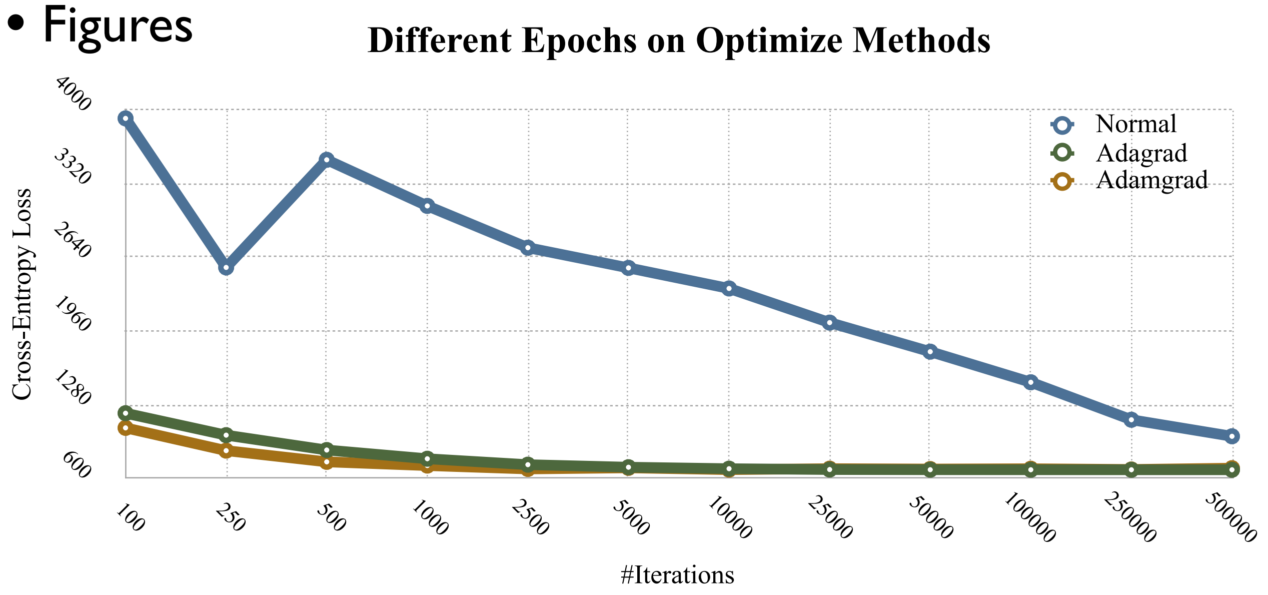


Figure 1.

Experiment results of different epochs on different optimize methods

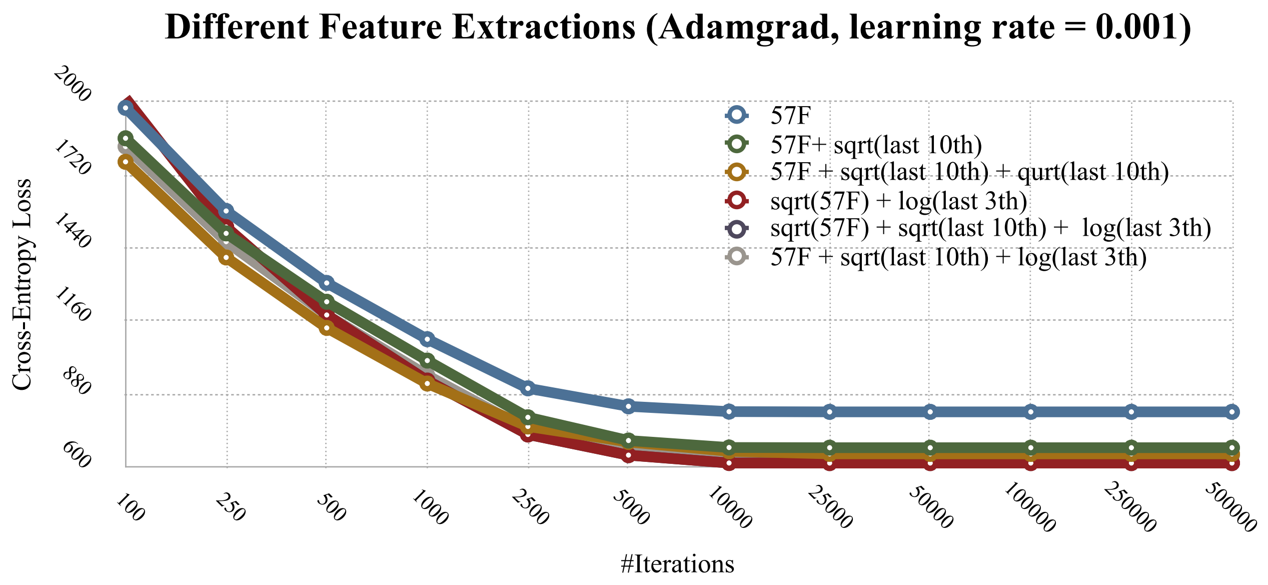


Figure 2.

Experiment results of different feature extraction method (F = feature, Adamgrad, learning rate = 0.001)

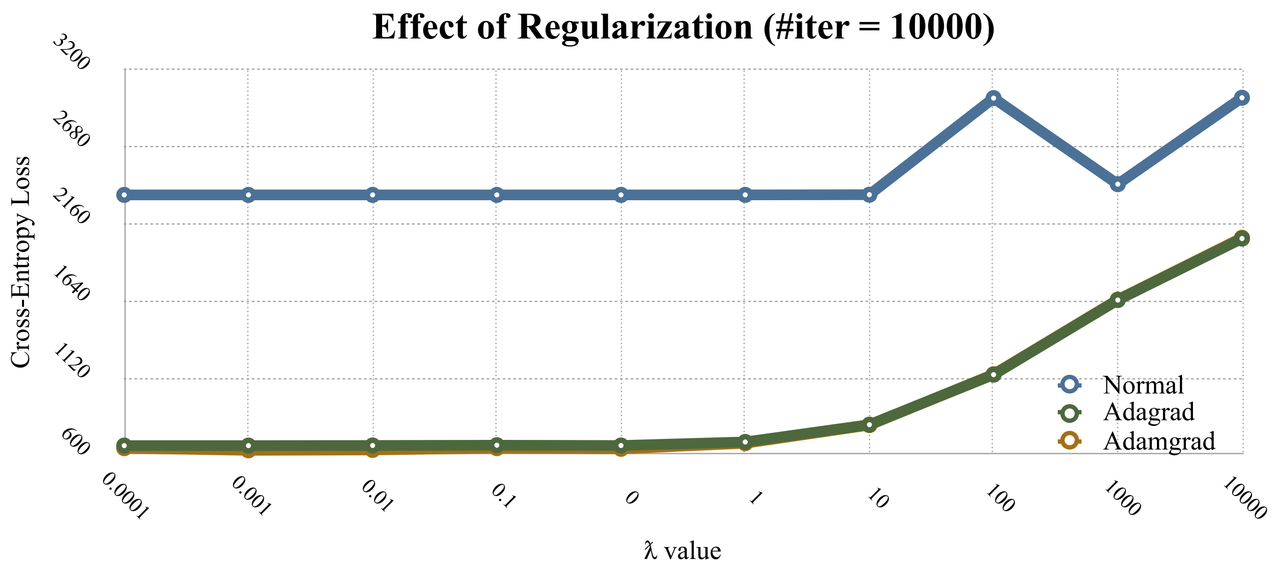
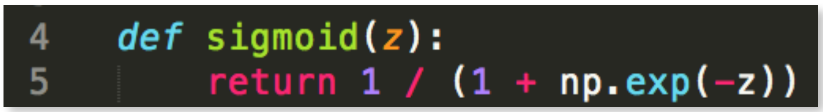


Figure 3.

Experiment results of the effect of regularization (#iter = 10000)

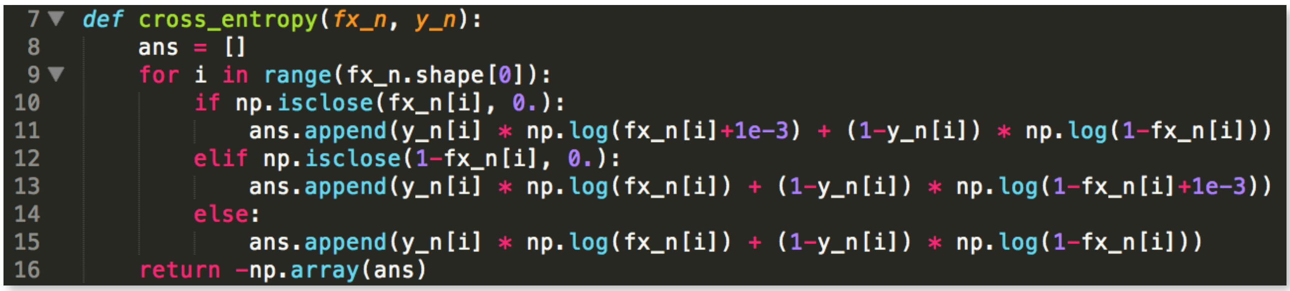
- Codes



```python
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

Figure. 4

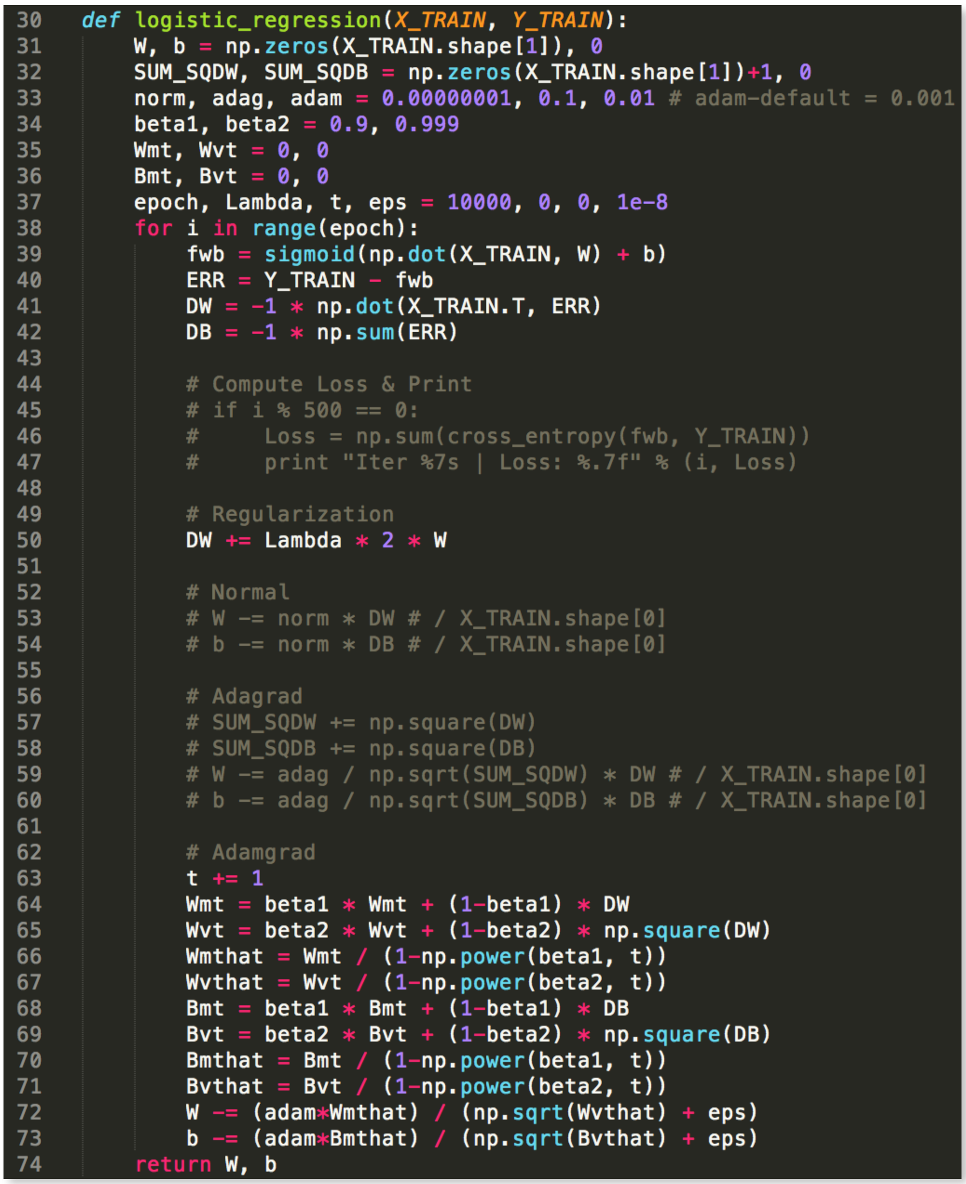
Sigmoid function implemented by numpy



```python
def cross_entropy(fx_n, y_n):
    ans = []
    for i in range(fx_n.shape[0]):
        if np.isclose(fx_n[i], 0.):
            ans.append(y_n[i] * np.log(fx_n[i]+1e-3) + (1-y_n[i]) * np.log(1-fx_n[i]))
        elif np.isclose(1-fx_n[i], 0.):
            ans.append(y_n[i] * np.log(fx_n[i]) + (1-y_n[i]) * np.log(1-fx_n[i]+1e-3))
        else:
            ans.append(y_n[i] * np.log(fx_n[i]) + (1-y_n[i]) * np.log(1-fx_n[i]))
    return -np.array(ans)
```

Figure. 5

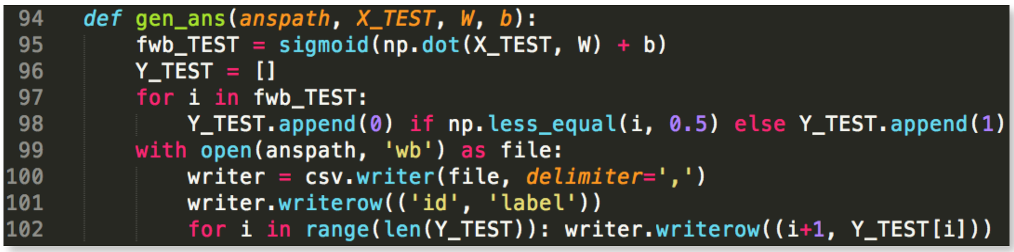
Compute the cross-entropy for each weight (they will be summed as the value of loss function)



```python
def logistic_regression(X_TRAIN, Y_TRAIN):
    W, b = np.zeros(X_TRAIN.shape[1]), 0
    SUM_SQDW, SUM_SQDB = np.zeros(X_TRAIN.shape[1])+1, 0
    norm, adag, adam = 0.00000001, 0.1, 0.01 # adam-default = 0.001
    beta1, beta2 = 0.9, 0.999
    Wmt, Wvt = 0, 0
    Bmt, Bvt = 0, 0
    epoch, Lambda, t, eps = 10000, 0, 0, 1e-8
    for i in range(epoch):
        fwb = sigmoid(np.dot(X_TRAIN, W) + b)
        ERR = Y_TRAIN - fwb
        DW = -1 * np.dot(X_TRAIN.T, ERR)
        DB = -1 * np.sum(ERR)

        # Compute Loss & Print
        # if i % 500 == 0:
        #     Loss = np.sum(cross_entropy(fwb, Y_TRAIN))
        #     print "Iter %7s | Loss: %.7f" % (i, Loss)

        # Regularization
        DW += Lambda * 2 * W

        # Normal
        # W -= norm * DW # / X_TRAIN.shape[0]
        # b -= norm * DB # / X_TRAIN.shape[0]

        # Adagrad
        # SUM_SQDW += np.square(DW)
        # SUM_SQDB += np.square(DB)
        # W -= adag / np.sqrt(SUM_SQDW) * DW # / X_TRAIN.shape[0]
        # b -= adag / np.sqrt(SUM_SQDB) * DB # / X_TRAIN.shape[0]

        # Adamgrad
        t += 1
        Wmt = beta1 * Wmt + (1-beta1) * DW
        Wvt = beta2 * Wvt + (1-beta2) * np.square(DW)
        Wmthat = Wmt / (1-np.power(beta1, t))
        Wvthat = Wvt / (1-np.power(beta2, t))
        Bmt = beta1 * Bmt + (1-beta1) * DB
        Bvt = beta2 * Bvt + (1-beta2) * np.square(DB)
        Bmthat = Bmt / (1-np.power(beta1, t))
        Bvthat = Bvt / (1-np.power(beta2, t))
        W -= (adam*Wmthat) / (np.sqrt(Wvthat) + eps)
        b -= (adam*Bmthat) / (np.sqrt(Bvthat) + eps)
    return W, b
```

Figure. 6

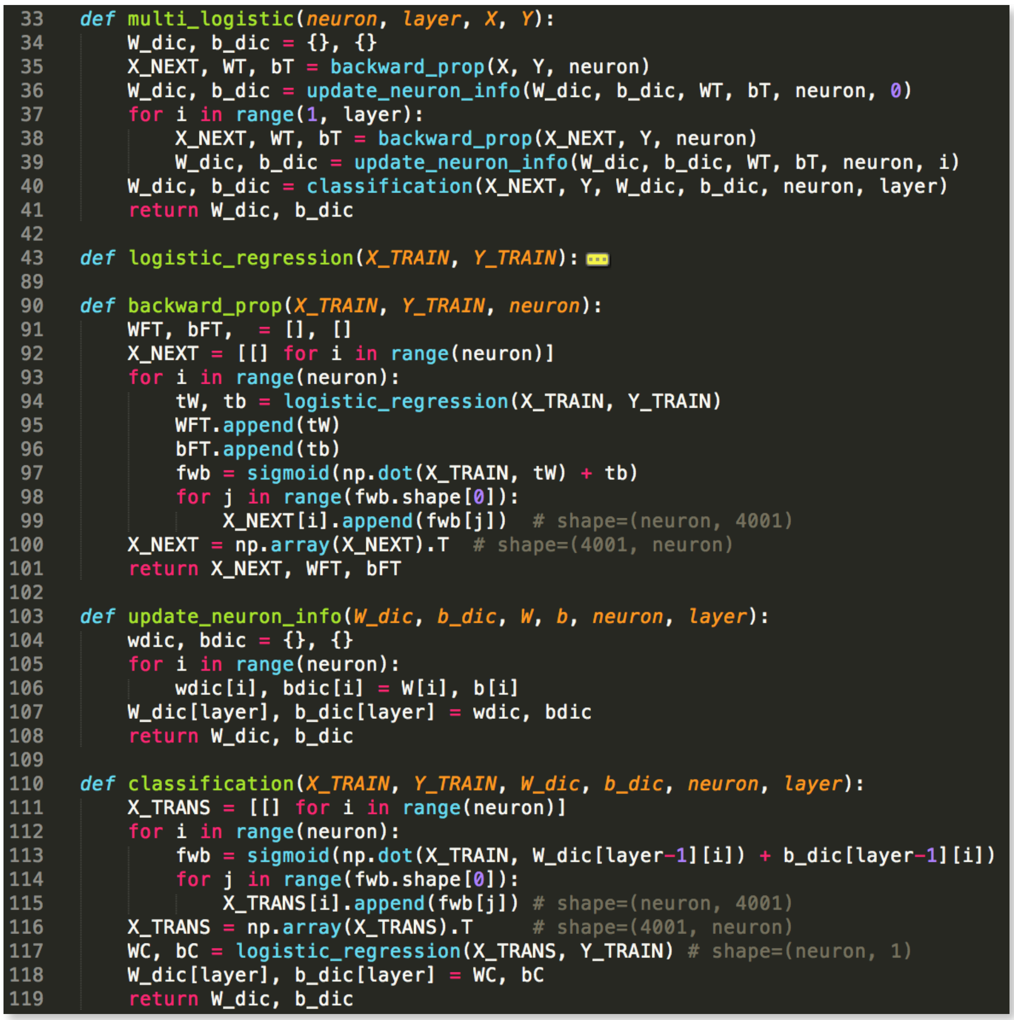
Logistic Regression (normal, adagrad, **adamgrad**)



```python
def gen_ans(anspath, X_TEST, W, b):
    fwb_TEST = sigmoid(np.dot(X_TEST, W) + b)
    Y_TEST = []
    for i in fwb_TEST:
        Y_TEST.append(0) if np.less_equal(i, 0.5) else Y_TEST.append(1)
    with open(anspath, 'wb') as file:
        writer = csv.writer(file, delimiter=',')
        writer.writerow(('id', 'label'))
        for i in range(len(Y_TEST)): writer.writerow((i+1, Y_TEST[i]))
```

Figure. 7

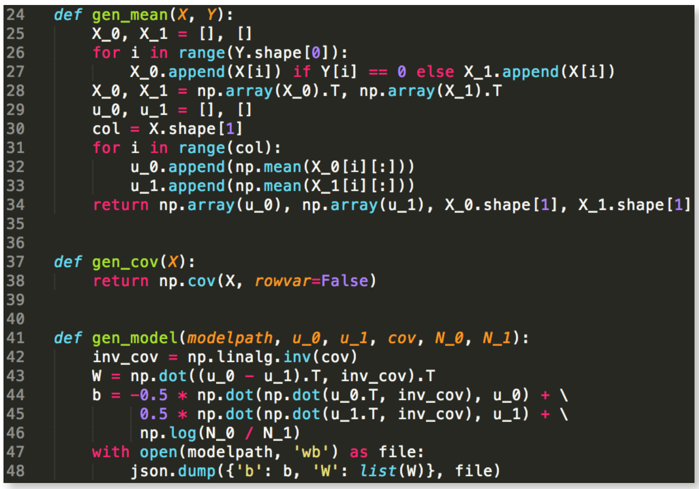
Generate the predicted results according to the probability (logistic)



```python
def multi_logistic(neuron, layer, X, Y):
    W_dic, b_dic = {}, {}
    X_NEXT, WT, bT = backward_prop(X, Y, neuron)
    W_dic, b_dic = update_neuron_info(W_dic, b_dic, WT, bT, neuron, 0)
    for i in range(1, layer):
        X_NEXT, WT, bT = backward_prop(X_NEXT, Y, neuron)
        W_dic, b_dic = update_neuron_info(W_dic, b_dic, WT, bT, neuron, i)
    W_dic, b_dic = classification(X_NEXT, Y, W_dic, b_dic, neuron, layer)
    return W_dic, b_dic

def logistic_regression(X_TRAIN, Y_TRAIN):

def backward_prop(X_TRAIN, Y_TRAIN, neuron):
    WFT, bFT,  = [], []
    X_NEXT = [[] for i in range(neuron)]
    for i in range(neuron):
        tW, tb = logistic_regression(X_TRAIN, Y_TRAIN)
        WFT.append(tW)
        bFT.append(tb)
        fwb = sigmoid(np.dot(X_TRAIN, tW) + tb)
        for j in range(fwb.shape[0]):
            X_NEXT[i].append(fwb[j])  # shape=(neuron, 4001)
    X_NEXT = np.array(X_NEXT).T  # shape=(4001, neuron)
    return X_NEXT, WFT, bFT

def update_neuron_info(W_dic, b_dic, W, b, neuron, layer):
    wdic, bdic = {}, {}
    for i in range(neuron):
        wdic[i], bdic[i] = W[i], b[i]
    W_dic[layer], b_dic[layer] = wdic, bdic
    return W_dic, b_dic

def classification(X_TRAIN, Y_TRAIN, W_dic, b_dic, neuron, layer):
    X_TRANS = [[] for i in range(neuron)]
    for i in range(neuron):
        fwb = sigmoid(np.dot(X_TRAIN, W_dic[layer-1][i]) + b_dic[layer-1][i])
        for j in range(fwb.shape[0]):
            X_TRANS[i].append(fwb[j]) # shape=(neuron, 4001)
    X_TRANS = np.array(X_TRANS).T      # shape=(4001, neuron)
    WC, bC = logistic_regression(X_TRANS, Y_TRAIN) # shape=(neuron, 1)
    W_dic[layer], b_dic[layer] = WC, bC
    return W_dic, b_dic
```

Figure. 8

Multi-Logistic Model (so called "Neural Network", normal, **adagrad**, adamgrad, #iter:7500)



```python
def gen_mean(X, Y):
    X_0, X_1 = [], []
    for i in range(Y.shape[0]):
        X_0.append(X[i]) if Y[i] == 0 else X_1.append(X[i])
    X_0, X_1 = np.array(X_0).T, np.array(X_1).T
    u_0, u_1 = [], []
    col = X.shape[1]
    for i in range(col):
        u_0.append(np.mean(X_0[i][:]))
        u_1.append(np.mean(X_1[i][:]))
    return np.array(u_0), np.array(u_1), X_0.shape[1], X_1.shape[1]


def gen_cov(X):
    return np.cov(X, rowvar=False)


def gen_model(modelpath, u_0, u_1, cov, N_0, N_1):
    inv_cov = np.linalg.inv(cov)
    W = np.dot((u_0 - u_1).T, inv_cov).T
    b = -0.5 * np.dot(np.dot(u_0.T, inv_cov), u_0) + \
        0.5 * np.dot(np.dot(u_1.T, inv_cov), u_1) + \
        np.log(N_0 / N_1)
    with open(modelpath, 'wb') as file:
        json.dump({'b': b, 'W': list(W)}, file)
```

Figure. 9

Probabilistic Generative Model