Machine Learning (2016 Fall) HW2 Report

Spam Classification by Logistic Regression

Student : 沈怡廷 r05943001

Kaggle_ID : r05943001_Panda

A. Logistic Regression Function

```python
""" logistic regression """
def logReg(X, y, X_test=None, y_test=None, lr=1e-7, batch=1, lamb=0,
           epoch=10000, print_every=100):
    # Initialize weight
    W = np.random.randn(X.shape[1]) / X.shape[1] / X.shape[0]
    L_train = []
    A_train = []
    L_test  = []
    A_test  = []

    # AdaGrad
    G = np.zeros(W.shape)

    for i in range(epoch):
        # batch
        b = 0
        idx = []

        for j in np.random.permutation(X.shape[0]):
            idx.append(j)
            b += 1

            if b >= batch:
                # prediction error
                pred_err = sigmoid(X[idx].dot(W)) - y[idx]

                # calculate gradient
                grad_X = X[idx].T.dot(pred_err)
                grad_regular = lamb * W * batch / X.shape[0]
                grad = grad_X + grad_regular

                # calculate weight
                G += grad**2
                W -= grad*lr/np.sqrt(G+1e-8)

                # reset parameter
                b = 0
                idx = []

        # prevent inf and nan
        cross_Entropy = croEntropy(sigmoid(X.dot(W)), y)
        cross_Entropy[np.isinf(cross_Entropy)] = 1000
        cross_Entropy[np.isnan(cross_Entropy)] = 0

        # calculate loss and accuracy
        Loss_train = cross_Entropy.sum() + lamb*(W**2).sum()
        Accuracy_train = calcAccuracy(X, W, y)

        L_train.append(Loss_train)
        A_train.append(Accuracy_train)

        # testing
        if X_test is not None and y_test is not None:
            cross_Entropy = croEntropy(sigmoid(X_test.dot(W)), y_test)
            cross_Entropy[np.isinf(cross_Entropy)] = 1000
            cross_Entropy[np.isnan(cross_Entropy)] = 0

            Loss_test = cross_Entropy.sum() + lamb*(W**2).sum()
            Accuracy_test = calcAccuracy(X_test, W, y_test)

            L_test.append(Loss_test)
            A_test.append(Accuracy_test)

        # print out the progress
        if i % print_every == 0:
            if X_test is not None and y_test is not None:
                print('\tepoch: %d; loss: %.4f; Acc_train: %.4f; Acc_test: %.4f' %
                      (i, Loss_train, Accuracy_train, Accuracy_test))
            else:
                print('\tepoch: %d; loss: %.4f; Acc_train: %.4f' %
                      (i, Loss_train, Accuracy_train))

    print('\nfinal loss: %.4f' % L_train[-1])
    print('final training accuracy: %.4f' % A_train[-1])
    print('final testing accuracy: %.4f' % A_test[-1])

    return W, L_train, A_train, L_test, A_test
```

B. Describe your another method, and which one is best

我實作的另外一個方法為「probabilistic generative model」，即上課所提到，假設資料分布呈現某種機率模型，利用 training data 的資料找出 average 及 variance，即可用來判斷 testing data。以下為其程式碼 ：

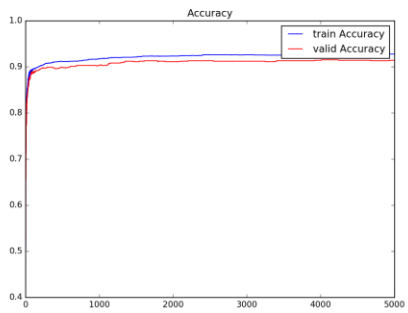
```python
""" generative model """
def probGen(X, y):
    data_class1 = [x for idx, x in enumerate(X) if y[idx]==1]
    data_class2 = [x for idx, x in enumerate(X) if y[idx]==0]

    # transform to numpy array
    data_class1_array = np.matrix(data_class1)
    data_class2_array = np.matrix(data_class2)

    # delete answer column
    data_class1_array = np.delete(data_class1_array, 57, 1)
    data_class2_array = np.delete(data_class2_array, 57, 1)

    # calculate number
    N1 = data_class1_array.shape[0]
    N2 = data_class2_array.shape[0]
    N  = N1 + N2

    # calculate mean
    u1 = np.mean(data_class1_array, axis=0)
    u2 = np.mean(data_class2_array, axis=0)

    # minus mean
    data_class1_array_zero_mean = data_class1_array - u1
    data_class2_array_zero_mean = data_class2_array - u2

    # square
    data_class1_square = data_class1_array_zero_mean.T*(data_class1_array_zero_mean)
    data_class2_square = data_class2_array_zero_mean.T*(data_class2_array_zero_mean)

    # variance
    var1 = data_class1_square/N1
    var2 = data_class2_square/N2
    var  = var1*N1/N + var2*N2/N

    # avoid singular
    var_mod = var + np.eye(var.shape[0])*1e-6
    var_inv = np.linalg.inv(var_mod)

    # weight
    W = (u1 - u2)*(var_inv)
    b = 0.5*u2*var_inv*(u2.T) - 0.5*u1*var_inv*(u1.T) + np.log(N1/N2)
    W = np.concatenate((W, b), axis=1)

    # accuracy
    Accuracy = calcAccuracy(X, W.T, y)

    return W.T, Accuracy
```

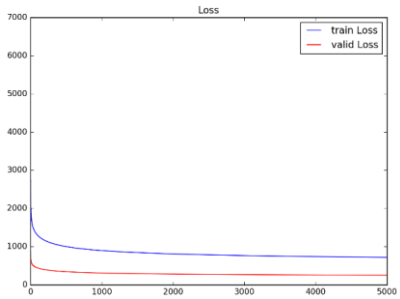
與 logistic regression 的比較如下圖所示，可以發現到，logistic regression 無論在 training data 或 testing data 上，表現均比較優異，主要原因或許是資料數量不夠多，機率模型有可能會有過多的錯誤判斷，導致效果較差。

	Training	Validation	Public Testing
Logistic	0.9280	0.9251	0.9333
Generative	0.8923	N/A	0.87667

C. TA depend on your other discussion and detail

a. Loss and accuracy during training





b. More about my logistic regression implementation

Initial weight and bias : random

Training set : random pick 3/4 of training data

Validation set : the rest 1/4 of training data

Optimization : AdaGrad

Batch size : whole training set (without SGD)

Learning rate : 0.01

Epoch : 5000

Regularization : None (lambda = 0)