HW 2 - Emma Young

Prediction using Bayes' theorem

1. Define discriminant functions using case 3 equation:

$$g_i(\mathbf{x}) = \mathbf{x}^t \mathbf{W}_i \mathbf{x} + \mathbf{w}_i^t \mathbf{x} + w_{i0},$$

$$\mathbf{W}_i = -\frac{1}{2}\boldsymbol{\Sigma}_i^{-1},$$

$$\mathbf{w}_i = \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i$$

$$w_{i0} = -\frac{1}{2}\boldsymbol{\mu}_i^t \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i - \frac{1}{2}\ln|\boldsymbol{\Sigma}_i| + \ln P(\omega_i).$$

There will be a discriminant function for every class, so we will create 10 functions (g0 to g9) to use later in classifying the testing data. Since the testing data is low rank from the amount of zeros it contains, we must take special care when taking the inverse and calculating the determinant of the covariance matrix. The pseudo inverse is calculated and an estimation of the determinant is found using SVD. This could be much more efficient by doing the inverse and determinant calculation once (outside the function).

```
def discriminant(x,COV,mu,Prior):
    #x must be a column vector
    #mu must be a column vector

    invCOV = LA.pinv(COV) #pseudo inverse
    [u, s, vh] = LA.svd(COV)

    s = s/s[0]
    detCOV = np.prod(s[s > 0.01])

    W = 0.5*invCOV
    w = invCOV@mu
    w0 = 0.5*np.transpose(mu)@invCOV@mu - 0.5*np.log(detCOV) + np.log(Prior)

    g = np.transpose(x)@W@x + np.transpose(w)@x + w0
    g = np.reshape(g, ())

    return g
```

2. Classify testing data

To save time, I used only the first 50 samples from each set of testing adata to test the accuracy of my model.

A vector "testCurr" (1x784) is passed into each discriminant function, and the resulting value is stored into a (1x10) vector called "gtest". The first column is the value of the first

discriminant function, the second is the value from the second discriminant function, and so on. This 1x10 vector is added into the 10x10 confusion matrix in the index of the row that corresponds to the ground truth of the training data. The gtest calculated from all of the train0 data would be put into the 1st row of the confusion matrix, the gtest calculated from all the train1 data would be put into the 2nd row of the confusion matrix, and so on. The confusion matrix is constructed in such a way that a true positive lies on the diagonal.

```
gtest[:,0] = discriminant(np.transpose(testCurr[i,:]), COV0, np.transpose(mean0), P0)
gtest[:,1] = discriminant(np.transpose(testCurr[i,:]), COV1, np.transpose(mean1), P1)
gtest[:,2] = discriminant(np.transpose(testCurr[i,:]), COV2, np.transpose(mean2), P2)
gtest[:,3] = discriminant(np.transpose(testCurr[i,:]), COV3, np.transpose(mean3), P3)
gtest[:,4] = discriminant(np.transpose(testCurr[i,:]), COV4, np.transpose(mean4), P4)
gtest[:,5] = discriminant(np.transpose(testCurr[i,:]), COV5, np.transpose(mean5), P5)
gtest[:,6] = discriminant(np.transpose(testCurr[i,:]), COV6, np.transpose(mean6), P6)
gtest[:,7] = discriminant(np.transpose(testCurr[i,:]), COV7, np.transpose(mean7), P7)
gtest[:,8] = discriminant(np.transpose(testCurr[i,:]), COV8, np.transpose(mean8), P8)
gtest[:,9] = discriminant(np.transpose(testCurr[i,:]), COV9, np.transpose(mean9), P9)
```

The Confusion matrix:

	0	1	2	3	4	5	6	7	8	9
0	49	0	0	0	0	1	0	0	0	0
1	0	2	3	0	8	0	0	0	37	0
2	1	0	45	1	0	0	0	1	2	0
3	0	0	4	40	1	1	0	1	3	0
4	0	0	1	0	45	0	1	0	2	1
5	2	0	1	0	0	41	1	0	4	1
6	2	0	1	0	0	3	44	0	0	0
7	0	0	0	2	0	0	0	46	2	0
8	1	0	3	2	1	1	0	0	42	0
9	0	0	1	1	4	0	0	2	3	39

With this confusion matrix, we can calculate the total classification accuracy for each class.

	Class Prediction Stats									
	0	1	2	3	4	5	6	7	8	9
#TP	49	2	45	40	45	41	44	46	42	39
#FN	1	48	5	10	5	9	6	4	8	11
#FP	6	0	14	6	14	6	2	4	53	2
#TN	444	450	436	444	436	444	448	446	397	448
Accuracy	0.986	0.904	0.962	0.968	0.962	0.970	0.984	0.984	0.878	0.974

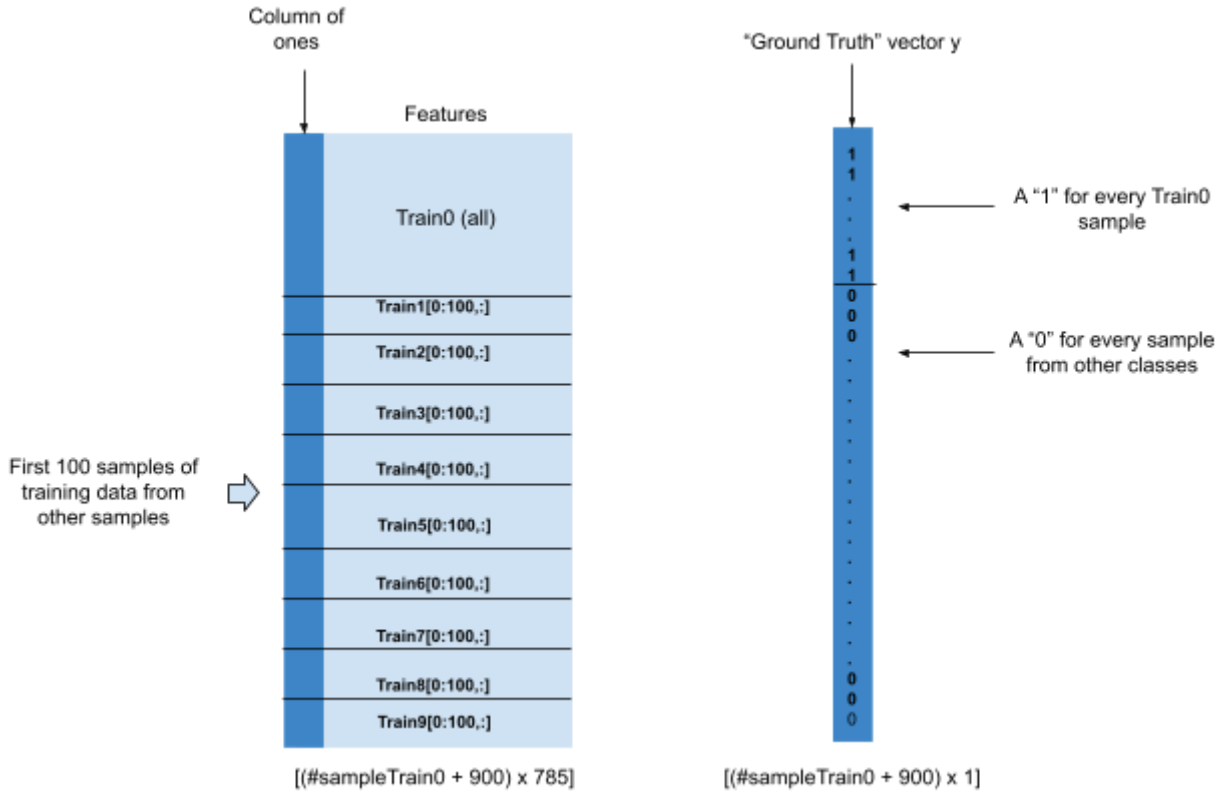
Prediction Using Logistic Regression

1. Prepare data to train a OnevsAll logistic regression model

A column of ones is added to every collection of training data, making the dimensions for each set:

[numSamples x numFeatures+1] = [numSamples x 785]

I will train ten OnevsAll models, and must prepare ten training samples with data for the desired classification, and data from the undesired classification. A ground truth vector is also created with the length of the number of samples in this prepared training data. This vector has a 1 corresponding with samples of the desired classification, and a 0 for the undesired classification. Here is an example for the data prepared to classify for the number “0”. Similar matrices and vectors are created for all 9 other classifiers.



2. Train regression using steepest descent of convex cost function to find “theta” (the weights for each logistic regression model).

Minimizing this cost function will be the minimization between the difference in predicted and ground truth classification. This is achieved by performing gradient descent on the regression weight vector “theta”

$$\text{Cost}(\boldsymbol{h_\theta}(\boldsymbol{x}), y) = -y \log\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right) - (1-y)\log\left(1 - \frac{1}{1+e^{-\boldsymbol{\theta}^T x}}\right)$$

This is achieved by performing gradient descent on the regression weight vector “theta”

$$\theta_j = \theta_j - \alpha \sum_{i=1}^{n} \left(\frac{1}{1+e^{-\boldsymbol{\theta}^T x^{(i)}}} - y^{(i)}\right) x_j{}^{(i)}$$

The final formula after applying partial derivatives

My gradient function:

```
def Gradient(theta, x, y):
    #y: ground truth, column vector
    # column of whatver the number actually should be

    #theta*x: prediction

    #x: input feature vector (testing data) COLUMN VECTOR (reshaped in loop)
    #theta: COLUMN VECTOR

    [numSample, numFeature] = np.shape(x)

    grad = np.zeros((numFeature,1))
    #grad = []
    for i in range(numSample):
        xvect = np.reshape(x[i,:], (numFeature,1)) # (785x1)
        predict = (1/(1-np.exp(-1*np.transpose(theta)@xvect)))
        grad = grad + (predict - y[i,:])*xvect #(785x1)



    return grad
```

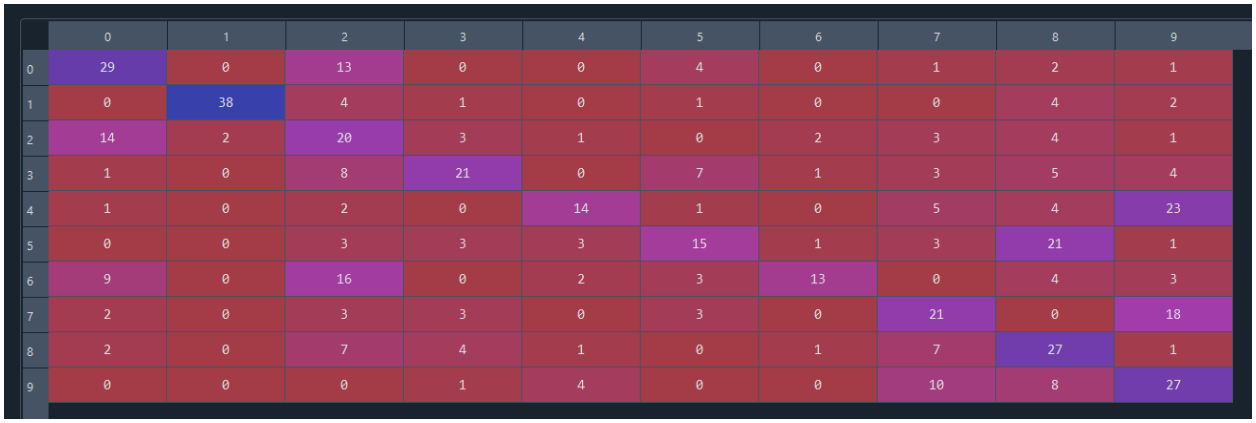
I create two functions to 1) Calculate the gradient with each iteration and 2) apply it in gradient descent. Theta is initialized as a column of ones, and is refined through 1000 iterations of the gradient descent algorithm with a learning rate of 0.001.

```
def gradDescent(x,y):
    print(gradDescent)
    theta = np.ones((np.shape(train0)[1],1))#np.random.normal
    #eps = 0.01 #stopping criteria
    #stopCrit = 100 #set large so it can enter the loop
    learnRate = 0.001
    #cost = []
    for i in range(1000):
        #calculate gradient
        gradient = Gradient(theta, x, y)
        #calculte cost function
        #cost.append(ObjFcn(theta, x, y))
        theta_p = theta - learnRate*gradient #(785x1)
        #print(learnRate*gradient)

        #stopCrit = LA.norm(theta - theta_p)
        theta = theta_p

    return theta
```

3. Test regression accuracy with a selection of training data from each class.

Each training sample is classified using sigmoid functions and the 10 weights calculated during gradient descent. The sigmoid function translates the regression score into a probability of classification. The testing sample will be classified using the regression that produces the highest probability of classificaton. If there are multiple classes with the highest probability, the class is randomly chosen amongst them.

Here is the confusion matrix after testing with 50 samples from each class.



	0	1	2	3	4	5	6	7	8	9
0	29	0	13	0	0	4	0	1	2	1
1	0	38	4	1	0	1	0	0	4	2
2	14	2	20	3	1	0	2	3	4	1
3	1	0	8	21	0	7	1	3	5	4
4	1	0	2	0	14	1	0	5	4	23
5	0	0	3	3	3	15	1	3	21	1
6	9	0	16	0	2	3	13	0	4	3
7	2	0	3	3	0	3	0	21	0	18
8	2	0	7	4	1	0	1	7	27	1
9	0	0	0	1	4	0	0	10	8	27

	Class Prediction Stats									
	0	1	2	3	4	5	6	7	8	9
#TP	29	38	20	21	14	15	13	21	27	27
#FN	21	12	30	29	36	35	37	29	23	23
#FP	29	2	56	15	11	19	5	32	52	54
#TN	421	448	394	435	439	431	445	418	398	396
Accuracy	0.90	0.972	0.828	0.912	0.906	0.892	0.916	0.878	0.850	0.846

The logistic regression model doesn't do as well of a job as the Bayes' classification. Next time, the selection of weights should be more refined.