AI61003 LINEAR ALGEBRA FOR AI AND ML
ASSIGNMENT 02 - PROBLEM 10

ANALYSIS ON TRAIN DATA

Confusion Matrix

The $10x10$ confusion matrix CM_{train} for the resultant model on the train set is as follows, where $CM_{train}[i][j]$ =number of images in the train set belonging to class i that were classified to the class j by the model.

```
[[972,   1,   3,   4,   2,   4,   8,   0,   5,   1],
 [  1, 971,   6,   1,   4,   5,   0,   0,  10,   2],
 [ 10,  44, 835,  25,  14,   3,  29,   7,  31,   2],
 [  2,  28,  27, 849,   7,  25,   6,  19,  11,  26],
 [  0,  16,   4,   1, 911,   7,   7,   6,   6,  42],
 [ 23,   8,   2,  58,  14, 811,  25,   5,  38,  16],
 [ 11,   5,  10,   0,   9,  17, 943,   0,   5,   0],
 [  7,  25,   4,   5,  26,   1,   0, 879,   0,  53],
 [ 14,  65,   8,  22,  15,  28,  17,   4, 801,  26],
 [  9,   9,   5,  15,  39,   2,   0,  58,   8, 855]]
```

Accuracy

The prediction accuracy of the model on the train set is 88.27%.

ANALYSIS ON TEST DATA

Confusion Matrix

The $10x10$ confusion matrix CM_{test} for the resultant model on the test set is as follows, where $CM_{test}[i][j]$ = number of images in the test set belonging to class i that were classified to the class j by the model.

```
[[95,  0,  1,  0,  0,  0,  3,  0,  1,  0],
 [ 0, 96,  0,  1,  0,  2,  0,  0,  1,  0],
 [ 1, 11, 66,  4,  1,  0,  3,  5,  9,  0],
 [ 0,  1,  2, 82,  2,  2,  2,  4,  2,  3],
 [ 0,  1,  0,  0, 88,  2,  1,  0,  2,  6],
 [ 4,  2,  0, 11,  3, 63,  2,  3,  9,  3],
 [ 3,  2,  1,  0,  3,  7, 83,  0,  1,  0],
 [ 0,  7,  1,  1,  3,  0,  0, 80,  0,  8],
 [ 0,  1,  4,  6,  8,  2,  1,  1, 76,  1],
 [ 0,  1,  0,  1, 12,  0,  0,  8,  3, 75]]
```

Accuracy

The prediction accuracy of the model on the test set is 80.40%.

In [1]:
```python
import tensorflow as tf
import numpy as np
```

In [2]:
```python
mnist_data = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist_data.load_data()
```

In [3]:
```python
# Generate train dataset of size 10000, with 1000 samples
# from every class from 0 to 9
train_data = []
counts = [ 0 ] * 10
for i in range(len(x_train)) :
    if len(train_data) == 10000 : break
    if counts[y_train[i]] == 1000 : continue
    counts[y_train[i]] += 1
    x = x_train[i].flatten() / 255
    train_data.append((x, y_train[i]))
```

In [4]:
```python
# Generate test dataset of size 1000, with 100 samples
# from every class from 0 to 9
test_data = []
counts = [ 0 ] * 10
for i in range(len(x_test)) :
    if len(test_data) == 1000 : break
    if counts[y_test[i]] == 100 : continue
    counts[y_test[i]] += 1
    x = x_test[i].flatten() / 255
    test_data.append((x, y_test[i]))
```

In [5]:
```python
# This function computes and returns the LS solution of Ax = b using
# Tikhonov's regularized inversion with a very small lambda, 0.000001.
# Without regularized inversion, a fatal error was returned because the
# matrix (A_t)A turned out to be non-invertible (singular).
# NOTE : A_t is transpose of A.
def Least_Squares_Solution ( A , b ) :
    t = np.matmul(np.transpose(A), A) + 0.000001 * np.eye(A.shape[1])
    t = np.linalg.inv(t)
    t = np.matmul(t, np.transpose(A))
    x_hat = np.matmul(t, b)
    return x_hat

# This function returns the Vandermonde matrix for the given data samples.
# The model chosen is linear in the features of the data samples. Since the
# data samples are 784-dimensional, there will be 785 basis functions -- one
# for each of the 784 features plus a basis function with constant value 1.
# For N training data samples, Vandermonde matrix will be of dimension Nx785.
def Vandermonde_Matrix ( data ) :
    van_mat = []
    for x, _ in data :
        f = [1] + list(x)
        van_mat.append(f)
    return np.array(van_mat)

# This function fits a binary classification model and returns the corresponding parameters.
# Each data sample belongs to either +1/"positive" class or -1/"negative" class.
def Linear_Binary_Classification_Model ( data ) :
    A = Vandermonde_Matrix(data)
    y = [ t for _, t in data ]
    return Least_Squares_Solution(A, y)
```

In [7]:
```python
# This function derives a multi-class classification model. Each data sample belongs
# to one of the 10 classes (namely, 0-9). Therefore, 10 binary classification models
# are trained, one for each of the class. Each binary model has 785 parameters, hence
# this multi-class classification model will have 785x10 = 7850 parameters.
def Linear_Multiclass_Classification_Model ( data ) :
    p = []
    for c in range(10) :
        data_c = [ ]
        for x, y in data :
            if y == c : data_c.append((x, 1))
            else : data_c.append((x, -1))
        p.append(Linear_Binary_Classification_Model(data_c))
    return p
```

In [8]:
```python
# This function predicts the class (from 0 to 9) to which the data sample x belongs, given
# the parameters p of the multi-class classification model.
def Predict_Class ( x , p ) :
    confidence = [ ] # confidence[c] is proportional to the confidence that x belongs
                     # to the class c. Finally, the class with the highest confidence
                     # is returned as the prediction of the model governed by p.
    for c in range(10) :
        X = np.array([1] + list(x)) # vector of values of the 785 basis functions
        pred_conf = np.dot(p[c], X)
        confidence.append(pred_conf)
    return np.argmax(confidence)

# This function constructs the confusion matrix for the given data samples with respect
# to the model defined by the given parameters p.
# If confusion matrix is cf_mat, cf_mat[c][pred] = the no. of data samples belonging to
# the true class c that were classified into the class pred by the trained model.
def Confusion_Matrix ( data , p ) :
    cf_mat = np.zeros((10, 10)).astype(np.int32)
    for x, y in data :
        cf_mat[y][Predict_Class(x, p)] += 1
    return cf_mat
```

In [9]:
```python
p = Linear_Multiclass_Classification_Model(train_data)
```

In [10]:
```python
# Confusion Matrix of the resultant model on the train data
cf_mat_train = Confusion_Matrix(train_data, p)
cf_mat_train
```

Out[10]:
```
array([[972,   1,   3,   4,   2,   4,   8,   0,   5,   1],
       [  1, 971,   6,   1,   4,   5,   0,   0,  10,   2],
       [ 10,  44, 835,  25,  14,   3,  29,   7,  31,   2],
       [  2,  28,  27, 849,   7,  25,   6,  19,  11,  26],
       [  0,  16,   4,   1, 911,   7,   7,   6,   6,  42],
       [ 23,   8,   2,  58,  14, 811,  25,   5,  38,  16],
       [ 11,   5,  10,   0,   9,  17, 943,   0,   5,   0],
       [  7,  25,   4,   5,  26,   1,   0, 879,   0,  53],
       [ 14,  65,   8,  22,  15,  28,  17,   4, 801,  26],
       [  9,   9,   5,  15,  39,   2,   0,  58,   8, 855]])
```

In [11]:
```python
# Prediction accuracy of the resultant model on the train data
acc_train = cf_mat_train.trace() * 100 / len(train_data)
print(' Train Accuracy :', round(acc_train, 3), '%')
```

```
 Train Accuracy : 88.27 %
```

In [12]:
```python
# Confusion Matrix of the resultant model on the test data
cf_mat_test = Confusion_Matrix(test_data, p)
cf_mat_test
```

Out[12]:
```
array([[95,  0,  1,  0,  0,  0,  3,  0,  1,  0],
       [ 0, 96,  0,  1,  0,  2,  0,  0,  1,  0],
       [ 1, 11, 66,  4,  1,  0,  3,  5,  9,  0],
       [ 0,  1,  2, 82,  2,  2,  2,  4,  2,  3],
       [ 0,  1,  0,  0, 88,  2,  1,  0,  2,  6],
       [ 4,  2,  0, 11,  3, 63,  2,  3,  9,  3],
       [ 3,  2,  1,  0,  3,  7, 83,  0,  1,  0],
       [ 0,  7,  1,  1,  3,  0,  0, 80,  0,  8],
       [ 0,  1,  4,  6,  8,  2,  1,  1, 76,  1],
       [ 0,  1,  0,  1, 12,  0,  0,  8,  3, 75]])
```

In [13]:
```python
# Prediction accuracy of the resultant model on the test data
acc_test = cf_mat_test.trace() * 100 / len(test_data)
print(' Test Accuracy :', round(acc_test, 3), '%')
```

```
 Test Accuracy : 80.4 %
```