CS 5783 - Machine Learning - Homework 2

Romit Maulik

September 2018

Question 1

The classification accuracy of a Naive-Bayes classifier was observed as 0.7226 (72.26%). The code for this classifier is appended to this document.

Question 2

The true and false positive rates for the mentioned test cases are shown in the following table:

Case	True positive Rate	False positive rate
Type I errors 5 times as costly	0.865	0.173
Type I errors 2 times as costly	0.865	0.182
Both errors equally costly	0.865	0.192
Type II errors 5 times as costly	0.875	0.192
Type II errors 2 times as costly	0.875	0.192

The ROC curve for this model is shown below:

Question 3

The average classification accuracies (after 5 fold validation) for the different nearest neighbor approaches are shown in the table below:

Number of neighbors	Average classification accuracy	Test Accuracy
1	0.845	0.853
3	0.85	0.853
5	0.857	0.853
7	0.843	0.853
9	0.825	0.853

An issue commonly seen was that the '7' and '1' labels were often confused due to non-straight 1s.

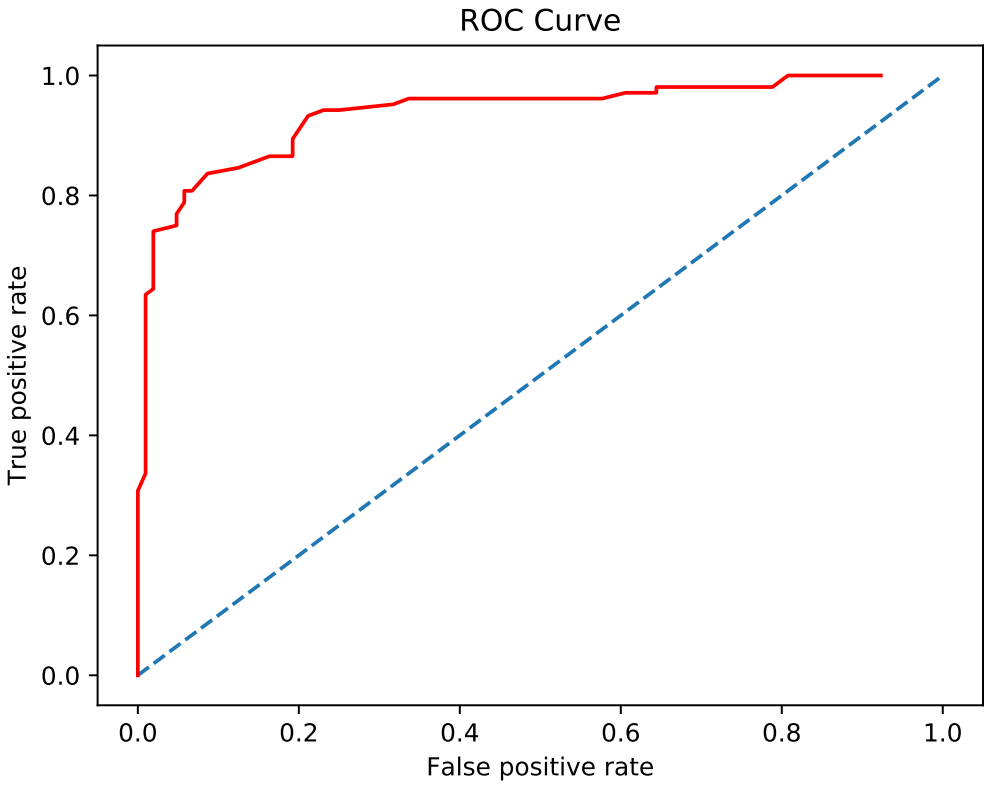

Figure 1: ROC curve for naive Gaussian Bayes classifier - Question 2

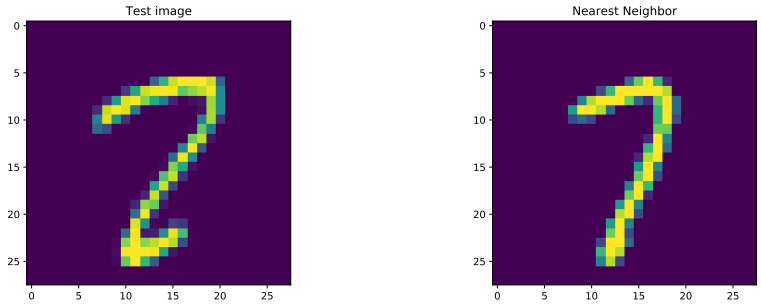

Figure 2: A mislabeled image: Note similarity between nearest neighbor (top) and test image (bottom). This test image was labeled a 1.

```
import numpy as np
import matplotlib.pyplot as plt
import os
import gzip as gz

def load_data(train_subset=12000,test_subset = 2000):
    #Training images
    data_file = gz.open(r'train-images-idx3-ubyte.gz','rb')
    training_images = data_file.read()
    data_file.close()
    training_images = bytearray(training_images)[16:]
    training_images = np.reshape(np.asarray(training_images),(60000,784))

    #Training labels
    data_file = gz.open(r'train-labels-idx1-ubyte.gz', 'rb')
    training_labels = data_file.read()
    data_file.close()
    training_labels = bytearray(training_labels)[8:]
    training_labels = np.reshape(np.asarray(training_labels),(60000,1))

    # Testing images
    data_file = gz.open(r't10k-images-idx3-ubyte.gz', 'rb')
    testing_images = data_file.read()
    data_file.close()
    testing_images = bytearray(testing_images)[16:]
    testing_images = np.reshape(np.asarray(testing_images), (10000, 784))

    # Testing labels
    data_file = gz.open(r't10k-labels-idx1-ubyte.gz', 'rb')
    testing_labels = data_file.read()
    data_file.close()
    testing_labels = bytearray(testing_labels)[8:]
    testing_labels = np.reshape(np.asarray(testing_labels),(10000,1))

    #Threshold training and test inputs
    training_images[training_images[:, :] < 128.0] = 0
    testing_images[testing_images[:, :] < 128.0] = 0
    training_images[training_images[:,:]>128.0] = 1
    testing_images[testing_images[:,:]>128.0] = 1


    #Random shuffling
    randomize = np.arange(np.shape(training_images)[0])
    np.random.shuffle(randomize)
    training_images = training_images[randomize]
    training_labels = training_labels[randomize]

    # Visualize
    # np.set_printoptions(threshold=np.nan)
    # print(training_images[0,:])
    #
    # check_row = np.reshape(training_images[0,:],(28,28))
    # plt.figure()
    # plt.imshow(check_row)
    # plt.show()

    return training_images[:train_subset,:], training_labels[:train_subset,:], testing_images[
:test_subset,:], testing_labels[:test_subset,:]

def naive_bayes_classifier(training_images,training_labels,testing_images,testing_labels):
    '''
    Note that there are 784 features in an input vector
    Need to calculate probability of each feature being in one class out of 10 for all trainin
```

```
g data
    '''
    dirichlet_prior = np.zeros(shape=(10),dtype='double')#Dirichlet prior
    for class_val in range(10):
        idx = np.ndarray.flatten(np.asarray((np.where(training_labels[:, 0] == class_val))))
        num_vals = np.shape(idx)[0]
        dirichlet_prior[class_val] = num_vals/np.shape(training_images)[0]

    #Choose a sample test image
    cond_probs = np.zeros(shape=(784,10),dtype='double')#Conditional probability of activated
pixel

    #Finding conditional probabilities
    for class_val in range(10):
        idx = np.ndarray.flatten(np.asarray((np.where(training_labels[:, 0] == class_val))))
        cond_probs[:,class_val] = np.count_nonzero(training_images[idx,:],axis=0)
        cond_probs[:,class_val] = cond_probs[:,class_val]/np.shape(training_images)[0]

    #Classifier
    label_preds = np.zeros(shape=(np.shape(testing_images)[0],1),dtype='int')
    correct = 0

    for k in range(np.shape(testing_images)[0]):
        class_probs = np.zeros(shape=(10), dtype='double')
        for class_val in range(10):
            class_probs[class_val] = dirichlet_prior[class_val] + np.sum(np.log(cond_probs[:,
class_val][np.where(testing_images[k, :] > 0)]))

        label_preds[k,0] = np.argmax(class_probs)

        if label_preds[k,0] == testing_labels[k,0]:
            correct = correct + 1

    print('Accuracy:',correct/np.shape(testing_images)[0])

#Main function
if __name__ == "__main__":
    training_images, training_labels, testing_images, testing_labels = load_data(60000,10000)
    naive_bayes_classifier(training_images,training_labels,testing_images,testing_labels)
```

```
import numpy as np
import matplotlib.pyplot as plt
import os
import gzip as gz


def load_data():

    np.random.seed(10)

    #Training images
    data_file = gz.open(r'train-images-idx3-ubyte.gz','rb')
    training_images = data_file.read()
    data_file.close()
    training_images = bytearray(training_images)[16:]
    training_images = np.reshape(np.asarray(training_images),(60000,784))

    #Training labels
    data_file = gz.open(r'train-labels-idx1-ubyte.gz', 'rb')
    training_labels = data_file.read()
    data_file.close()
    training_labels = bytearray(training_labels)[8:]
    training_labels = np.reshape(np.asarray(training_labels),(60000,1))

    # Testing images
    data_file = gz.open(r't10k-images-idx3-ubyte.gz', 'rb')
    testing_images = data_file.read()
    data_file.close()
    testing_images = bytearray(testing_images)[16:]
    testing_images = np.reshape(np.asarray(testing_images), (10000, 784))

    # Testing labels
    data_file = gz.open(r't10k-labels-idx1-ubyte.gz', 'rb')
    testing_labels = data_file.read()
    data_file.close()
    testing_labels = bytearray(testing_labels)[8:]

    #Choose 1000 samples of 5s
    idx = np.ndarray.flatten(np.asarray((np.where(training_labels[:, 0] == 5))))
    idx = np.random.choice(idx, 1000, replace=False)
    five_images = training_images[idx,:]
    five_labels = training_labels[idx,:]
    five_means = np.mean(five_images,axis=0)
    five_vars = np.var(five_images)

    #Choose 1000 samples of not 5s
    idx = np.ndarray.flatten(np.asarray(np.where(training_labels[:, 0] != 5)))
    idx = np.random.choice(idx, 1000, replace=False)
    not_five_images = training_images[idx,:]
    not_five_labels = training_labels[idx, :]
    not_five_means = np.mean(not_five_images,axis=0)
    not_five_vars = np.var(not_five_images)

    images = np.concatenate((five_images,not_five_images),axis=0)
    labels = np.concatenate((five_labels, not_five_labels), axis=0)

    idx = np.random.choice(np.arange(0,2000,1),2000,replace=False)

    testing_images = images[idx, :][1800:, :]
    testing_labels = labels[idx, :][1800:, :]

    # Visualize
    # np.set_printoptions(threshold=np.nan)
```

```
    # print(training_images[0,:])
    #
    # check_row = np.reshape(training_images[10,:],(28,28))
    # plt.figure()
    # plt.imshow(check_row)
    # plt.show()
    # print(training_labels[10,:])

    return testing_images, testing_labels, [five_means, not_five_means, five_vars, not_five_va
rs]

def gaussian_bayes_classifier(testing_images,testing_labels,parameters,tau):
    '''
        Note that there are 784 features in an input vector
        Each feature will have a mean given by a Gaussian conditional probability when 5
        Each feature will have a different mean given by a different Gaussian conditional prob
ability when not 5
        All features have same variance when class is 5
        All features have same variance when class is not 5
    '''
    five_means = parameters[0]
    not_five_means = parameters[1]
    five_vars = parameters[2]
    not_five_vars = parameters[3]

    # Classifier
    label_preds = np.zeros(shape=(np.shape(testing_images)[0], 1), dtype='int')
    correct = 0

    tp = 0
    fp = 0
    tn = 0
    fn = 0
    pos = 0
    neg = 0

    for k in range(np.shape(testing_images)[0]):
        class_probs = np.zeros(shape=(2), dtype='double')

        class_probs[1] = np.sum(np.log(1.0 / np.sqrt(2.0 * np.pi * five_vars) * np.exp(
                -((testing_images[k, :] - five_means[:]) ** 2) / (2.0 * five_vars))))

        class_probs[0] = np.sum(np.log(1.0 / np.sqrt(2.0 * np.pi * not_five_vars) * np.exp(
            -((testing_images[k, :] - not_five_means[:]) ** 2) / (2.0 * not_five_vars))))

        dec_prob = class_probs[1]-class_probs[0]

        if dec_prob >= tau:
            label_preds[k, 0] = 1
        else:
            label_preds[k, 0] = 0

        if testing_labels[k,0] == 5 and label_preds[k,0] == 1:
            correct = correct + 1
            tp = tp + 1
            pos = pos + 1
        elif testing_labels[k,0] != 5 and label_preds[k,0] == 0:
            correct = correct + 1
            tn = tn + 1
            neg = neg + 1
        elif testing_labels[k,0] != 5 and label_preds[k,0] == 1:
            fp = fp + 1
            neg = neg + 1
```

```
        elif testing_labels[k,0] == 5 and label_preds[k,0] == 0:
            fn = fn + 1
            pos = pos + 1

    #print('Accuracy:', correct / 200)
    #print('False positive rate:', fp / pos)
    #print('True positive rate:', tp / pos)

    return fp/pos, tp/pos


if __name__ == "__main__":
    testing_images, testing_labels, parameters = load_data()

    #Plotting ROC curve
    fig, ax = plt.subplots(nrows=1,ncols=1)
    ax.set_title('ROC Curve')
    ax.set_xlabel('False positive rate')
    ax.set_ylabel('True positive rate')

    roc_vals = np.empty(shape=(0,2),dtype='double')
    #gaussian_bayes_classifier(testing_images, testing_labels, parameters, np.log(1 / 1))
    #exit()

    print('Type 1 errors five times as costly: FPR,TPR = ',gaussian_bayes_classifier(testing_i
mages,testing_labels,parameters,np.log(5/1)))
    print('Type 1 errors two times as costly: FPR,TPR = ',gaussian_bayes_classifier(testing_im
ages,testing_labels,parameters,np.log(2/1)))
    print('Type 1 errors equally as costly: FPR,TPR = ',gaussian_bayes_classifier(testing_imag
es,testing_labels,parameters,np.log(1/1)))
    print('Type 2 errors two times as costly: FPR,TPR = ',gaussian_bayes_classifier(testing_im
ages,testing_labels,parameters,np.log(1/2)))
    print('Type 2 errors five times as costly: FPR,TPR = ',gaussian_bayes_classifier(testing_i
mages,testing_labels,parameters,np.log(1/5)))

    for tau in range(200,-200,-2):
        fp, tp = gaussian_bayes_classifier(testing_images,testing_labels,parameters,tau)
        roc_vals = np.append(roc_vals,np.array([[fp, tp]]),axis=0)

    sline = np.zeros(shape=(2,2),dtype='double')
    sline[0,0] = 0.0
    sline[0,1] = 0.0
    sline[1,0] = 1.0
    sline[1,1] = 1.0

    ax.plot(sline[:,0],sline[:,1],linestyle='dashed')
    ax.plot(roc_vals[:,0],roc_vals[:,1],color='red')
    plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
import gzip as gz


def load_data():

    np.random.seed(2)

    #Training images
    data_file = gz.open(r'train-images-idx3-ubyte.gz','rb')
    training_images = data_file.read()
    data_file.close()
    training_images = bytearray(training_images)[16:]
    training_images = np.reshape(np.asarray(training_images),(60000,784))

    #Training labels
    data_file = gz.open(r'train-labels-idx1-ubyte.gz', 'rb')
    training_labels = data_file.read()
    data_file.close()
    training_labels = bytearray(training_labels)[8:]
    training_labels = np.reshape(np.asarray(training_labels),(60000,1))

    # Testing images
    data_file = gz.open(r't10k-images-idx3-ubyte.gz', 'rb')
    testing_images = data_file.read()
    data_file.close()
    testing_images = bytearray(testing_images)[16:]
    testing_images = np.reshape(np.asarray(testing_images), (10000, 784))

    # Testing labels
    data_file = gz.open(r't10k-labels-idx1-ubyte.gz', 'rb')
    testing_labels = data_file.read()
    data_file.close()
    testing_labels = bytearray(testing_labels)[8:]
    testing_labels = np.reshape(np.asarray(testing_labels),(10000,1))


    #Creating training data
    idx1 = np.ndarray.flatten(np.asarray((np.where(training_labels[:, 0] == 1))))
    idx2 = np.ndarray.flatten(np.asarray((np.where(training_labels[:, 0] == 2))))
    idx7 = np.ndarray.flatten(np.asarray((np.where(training_labels[:, 0] == 7))))
    idx1 = np.random.choice(idx1, 200, replace=False)
    idx2 = np.random.choice(idx2, 200, replace=False)
    idx7 = np.random.choice(idx7, 200, replace=False)

    idx = np.concatenate((idx1, idx2, idx7), axis=0)
    training_images = training_images[idx, :]
    training_labels = training_labels[idx, :]

    #Creating testing data
    idx1 = np.ndarray.flatten(np.asarray((np.where(testing_labels[:, 0] == 1))))
    idx2 = np.ndarray.flatten(np.asarray((np.where(testing_labels[:, 0] == 2))))
    idx7 = np.ndarray.flatten(np.asarray((np.where(testing_labels[:, 0] == 7))))
    idx1 = np.random.choice(idx1, 50, replace=False)
    idx2 = np.random.choice(idx2, 50, replace=False)
    idx7 = np.random.choice(idx7, 50, replace=False)

    idx = np.concatenate((idx1, idx2, idx7), axis=0)
    testing_images = testing_images[idx, :]
    testing_labels = testing_labels[idx, :]

    return training_images, training_labels, testing_images, testing_labels
```

```
def distance_metric(sample,data):
    '''
    :param sample: An image feature vector to be classified
    :param data:  Training data of many feature vectors
    :return; set of distances
    '''
    ret_mat = np.zeros(shape=(np.shape(data)[0],),dtype='double')
    for i in range(np.shape(data)[0]):
        training_data = data[i,:]
        ret_mat[i] = np.sum(np.square(1.0*(sample-training_data)))

    return ret_mat

def classify_brute_force(training_images,training_labels,testing_images, testing_labels,k):
    '''
    :param testing_images: Image feature vectors to be classified
    :param training_images: Training data of many feature vectors
    :param training_labels: Training labels of many feature vectors
    :param testing_labels: Testing labels of many feature vectors for accuracy assessment
    :param k: Number of nearest neighbors
    :return:
    '''

    correct = 0
    for i in range(np.shape(testing_images)[0]):
        sample = testing_images[i,:]

        row_id = distance_metric(sample,training_images).argsort()[0:k]
        pred_label = training_labels[row_id,0]

        classification = np.argmax([np.sum(pred_label==1), np.sum(pred_label==2), np.sum(pred_
label==7)])

        if classification == 0:
            prediction = 1
        elif classification == 1:
            prediction = 2
        else:
            prediction = 7

        true_label = testing_labels[i,0]

        if prediction == true_label:
            correct = correct + 1

    #print('Classification accuracy: ',correct/np.shape(testing_images)[0])

    return correct/np.shape(testing_images)[0]

def classify_brute_force_test(training_images,training_labels,testing_images, testing_labels,k
,disp):
    '''
    :param testing_images: Image feature vectors to be classified
    :param training_images: Training data of many feature vectors
    :param training_labels: Training labels of many feature vectors
    :param testing_labels: Testing labels of many feature vectors for accuracy assessment
    :param k: Number of nearest neighbors
    :return:
    '''

    correct = 0
    for i in range(np.shape(testing_images)[0]):
```

```
        sample = testing_images[i,:]
        row_id = distance_metric(sample,training_images).argsort(axis=0)[0:k]

        pred_label = training_labels[row_id[0],0]
        classification = np.argmax([np.sum(pred_label==1), np.sum(pred_label==2), np.sum(pred_
label==7)])

        if classification == 0:
            prediction = 1
        elif classification == 1:
            prediction = 2
        else:
            prediction = 7

        true_label = testing_labels[i,0]

        if prediction == true_label:
            correct = correct + 1
        elif disp == True:
            # Visualize
            check_row_nn = np.reshape(training_images[row_id[0],:],(28,28))
            check_row_test = np.reshape(testing_images[i, :], (28, 28))

            plt.figure()
            plt.imshow(check_row_test)
            plt.title('Test image')
            plt.show()

            plt.figure()
            plt.imshow(check_row_nn)
            plt.title('Nearest Neighbor')
            plt.show()


    print('Classification accuracy: ',correct/np.shape(testing_images)[0])

def k_fold_cv(training_images,training_labels,n_folds):
    fold_size = int(np.shape(training_images)[0]/n_folds)
    for num_neighbors in range(1,10,2):

        average_accuracy = 0.0
        for fold in range(n_folds):
            start_ind = int(fold*fold_size)
            end_ind = int((fold+1)*fold_size)

            fold_testing_images = training_images[start_ind:end_ind,:]
            fold_testing_labels = training_labels[start_ind:end_ind, :]

            fold_training_images = np.delete(training_images, np.arange(start_ind,end_ind),0)
            fold_training_labels = np.delete(training_labels, np.arange(start_ind,end_ind),0)

            accuracy = classify_brute_force(fold_training_images, fold_training_labels, fold_t
esting_images, fold_testing_labels, num_neighbors)

            average_accuracy = average_accuracy + accuracy

        average_accuracy = average_accuracy/n_folds
        print('For ', num_neighbors,' neighbors, the average accuracy is: ',average_accuracy)
8
if __name__ == "__main__":
    training_images, training_labels, testing_images, testing_labels = load_data()
    #k_fold_cv(training_images,training_labels,5)
    classify_brute_force_test(training_images,training_labels,testing_images,testing_labels,1,
```

```
disp=True)
```