Name: Pham Minh Nghia

SID: 32507133

Problem 1:

1. Because there are 50 ‘5’s and 50 ‘9’s, the accuracy of an empty decision tree on the training set is 50%. If the training set is a random set then the accuracy is number\_of\_majority/total. On this particular test set, accuracy is 50% with the same above explanation.

Code:

#This code is part of:  
#  
# CMPSCI 370: Computer Vision, Spring 2021  
# University of Massachusetts, Amherst  
# Instructor: Subhransu Maji  
#  
# Mini-Project 6  
  
import pickle  
import numpy as np  
import matplotlib.pyplot as plt  
  
  
def scoreFeatures(x, y):  
 scores = np.zeros(x.shape[:2])  
 five, nine = np.zeros(x.shape[:2]), np.zeros(x.shape[:2])  
 #--------------------------------------------------------------------------  
 # Calculate scores (Implement this)  
 #--------------------------------------------------------------------------  
  
 # Calculate each attribute  
 for i in range(len(x[:, :, 1])): # loop through each pixel feature  
 for j in range(len(x[i, :, 1])):  
 for img in range(len(x[i, j])): # loop through each image  
 if x[i, j, img] == 0: # 1 is on and 0 is off  
 if y[img] == 5: # 5 is 'hated'  
 five[i, j] += 1  
 else:  
 if y[img] == 9: # 9 is 'liked'  
 nine[i, j] += 1  
  
 scores = five + nine  
 plt.imshow(scores, cmap='gray')  
 plt.axis('off')  
 plt.show()  
 return scores  
  
  
def highest\_score(scores):  
 r\_i, r\_j = 0, 0  
 highest = 0  
 for i in range(scores.shape[0]):  
 for j in range(scores.shape[1]):  
 check = highest  
 highest = max(highest, scores[i, j])  
 if highest != check:  
 r\_i, r\_j = i, j  
 print(r\_i, r\_j, highest)  
 return r\_i, r\_j  
  
  
def decisiontree\_test(x, y, i, j):  
 accuracy = 0  
 predict = []  
 for img in range(len(x[i, j])): # loop through each image  
 if x[i, j, img] == 0:  
 predict.append(5)  
 else:  
 predict.append(9)  
  
 for k in range(len(predict)):  
 if predict[k] == y[k]:  
 accuracy += 1  
 print("Accuracy =", accuracy/len(x[i, j]) \* 100, '%')  
 return None  
def depth2(x, y, x\_test, y\_test):  
 scores\_p = scoreFeatures(x, y)  
 i\_p, j\_p = highest\_score(scores\_p)  
 sub\_l, sub\_r = [], [] # Assign left dataset and right dataset  
 label\_l, label\_r = [], [] # left and right label  
  
 for img in range(len(x[i\_p, j\_p])): # loop through each image  
 if x[i\_p, j\_p, img] == 0: # pixel=0 go to left  
 sub\_l.append(x[:, :, img])  
 label\_l.append(y[img])  
 else: # pixel=1 go to right  
 sub\_r.append(x[:, :, img])  
 label\_r.append(y[img])  
  
 # convert to numpy array  
 sub\_l = np.array(sub\_l)  
 sub\_r = np.array(sub\_r)  
  
 # calculate scores for left and right datasets  
 scores\_l = scoreFeatures(sub\_l.transpose(1, 2, 0), label\_l)  
 scores\_r = scoreFeatures(sub\_r.transpose(1, 2, 0), label\_r)  
 i\_l, j\_l = highest\_score(scores\_l)  
 i\_r, j\_r = highest\_score(scores\_r)  
  
 # Prediction  
 predict = []  
 for img in range(len(x\_test[i\_p, j\_p])): # loop through each image  
 if x\_test[i\_p, j\_p, img] == 0:  
 if x\_test[i\_l, j\_l, img] == 0:  
 predict.append(5)  
 else:  
 predict.append(9)  
 else:  
 if x\_test[i\_r, j\_r, img] == 0:  
 predict.append(5)  
 else:  
 predict.append(9)  
  
 # Calculate accuracy  
 accuracy = 0  
 for i in range(len(predict)):  
 if predict[i] == y\_test[i]:  
 accuracy += 1  
 print("Accuracy =", accuracy / len(x\_test[i\_p, j\_p]) \* 100, '%')  
  
  
  
def main():  
 data = pickle.load(open('data\_binarized.pkl', 'rb'))  
 scores = scoreFeatures(data['train']['x'], data['train']['y'])  
 i, j = highest\_score(scores)  
 decisiontree\_test(data['test']['x'], data['test']['y'], i, j)  
  
 depth2(data['train']['x'], data['train']['y'], data['test']['x'], data['test']['y'])  
 #--------------------------------------------------------------------------  
 # Your implementation to answer questions on Decision Trees  
 #--------------------------------------------------------------------------  
  
 return  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Result:

A picture containing graphical user interface

Description automatically generated

Depth = 1

Highest Value: [11 18]= 163.0

Accuracy = 82.5 %

Depth = 2

Parent: [11 18]= 163.0

Left: [12 20]= 100.0

Right: [13 17]= 74.0

Graphical user interface, text

Description automatically generated with medium confidenceAccuracy = 90.5 %

Icon

Description automatically generated with medium confidence

Problem 2:

Code:

#This code is part of:  
#  
# CMPSCI 370: Computer Vision, Spring 2021  
# University of Massachusetts, Amherst  
# Instructor: Subhransu Maji  
#  
# Mini-Project 6  
  
import pickle  
import numpy as np  
import matplotlib.pyplot as plt  
  
  
def visualizeNeighbors(imgs, topk\_idxs, topk\_distances, title):  
 *'''  
 Visualize the query image as well as its nearest neighbors  
 Input:  
 imgs: a list or numpy array, with length k+1.  
 imgs[0] is the query image with shape hxw  
 imgs[k] is k-th nearest neighbor image  
 topk\_idxs: a list or numpy array, with length k+1.  
 topk\_idxs[k] is the index in training set of the k-th nearest image   
 topk\_idxs[0] is the query image index in the test set  
 topk\_distances: a list or numpy array, with length k+1.  
 topk\_idxs[k] is the distance of the k-th nearest image to the query  
 topk\_idxs[0] is 0  
 '''* n = len(imgs)  
 fig, axs = plt.subplots(1, n, figsize=(2 \* n, 3))  
 fig.suptitle(title)  
 for k in range(n):  
 if k == 0:  
 ax\_title = 'query: test\_idx=%d' % topk\_idxs[0]  
 else:  
 ax\_title = '%d: idx=%d,d=%.2e' %(k, topk\_idxs[k], topk\_distances[k])  
 axs[k].set\_title(ax\_title)  
 axs[k].imshow(imgs[k], cmap='gray')  
 axs[k].axis('off')  
 fig.tight\_layout()  
 plt.show()   
  
 return   
  
  
def knn(visualize):  
 data = pickle.load(open('data\_binarized.pkl','rb'))  
  
 # distances = np.zeros((len(data['test']['y']), len(data['train']['y'])))  
  
 # List of dictionary  
 # element i-th is a dictionary with key [i-th, train img]  
 # dict keys: [test img, train img]  
 # dict value: Euclidean distance  
 distances = [{}] \* 200  
 #--------------------------------------------------------------------------  
 # Your implementation to calculate and sort distances  
 #--------------------------------------------------------------------------  
 x\_train = data['train']['x']  
 x\_test = data['test']['x']  
 for i in range(len(data['test']['y'])): # loop through each image  
 diction = {}  
 for j in range(len(data['train']['y'])):  
 # Apply Euclidean formula  
 diction[i, j] = np.sqrt(np.sum(np.square(x\_test[:, :, i] - x\_train[:, :, j])))  
 # Sort dictionary by values  
 diction = dict(sorted(diction.items(), key=lambda item: item[1]))  
 distances[i] = diction  
  
 if visualize:  
 k = 5  
 imgs = np.random.randint(2, size=(k+1, 28, 28))  
 topk\_idxs = [0] \* (k+1)  
 topk\_distances = [0] \* (k+1)  
 for test\_i in [10, 20, 110, 120]:  
 topk\_idxs[0] = test\_i  
 # Assign query img as test\_i  
 imgs[0] = x\_test[:, :, test\_i]  
 #------------------------------------------------------------------  
 # Prepare imgs, topk\_idxs and topk\_distances  
 #------------------------------------------------------------------  
 i = 0  
 for key in distances[test\_i]:  
 i += 1 # Only go through k nearest neighbor  
 imgs[i] = x\_test[:, :, key[1]] # key[1] is index for training img of k-th neighbor  
 topk\_idxs[i] = key[1]  
 topk\_distances[i] = distances[test\_i][key]  
 if i >= k:  
 break  
  
 visualizeNeighbors(imgs, topk\_idxs, topk\_distances,   
 title='Test img %d: Top %d Neighbors' % (test\_i, k))  
  
 k\_list = [1, 3, 5, 7, 9]  
 accuracy\_list = [0.0] \* len(k\_list)  
 #--------------------------------------------------------------------------  
 # Your implementation to calculate knn accuracy  
 #--------------------------------------------------------------------------  
 for k, acc in zip(k\_list, accuracy\_list):  
 # Go through all test imgages  
 for i in range(len(data['test']['y'])):  
 five, nine, j = 0, 0, 0  
 for key in distances[i]:  
 j += 1 # Only check k nearest neighbor  
  
 # Finding majority  
 if data['train']['y'][key[1]] == 5:  
 five += 1  
 else:  
 nine += 1  
 if j >= k:  
 break  
  
 if five >= nine: # 5 is majority  
 if data['test']['y'][i] == 5:  
 acc += 1  
 else: # 9 is majority  
 if data['test']['y'][i] == 9:  
 acc += 1  
 # divide by total image (200) to get probability  
 acc /= len(data['test']['y'])  
 print('k=%d: accuracy=%.2f%%' % (k, acc \* 100))  
  
 return  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 knn(visualize=True)

Result:

k=1: accuracy=98.00%

k=3: accuracy=97.50%

k=5: accuracy=97.50%

k=7: accuracy=97.50%

k=9: accuracy=98.00%

Text

Description automatically generated with low confidence

Text, qr code

Description automatically generated

Qr code

Description automatically generated

Qr code

Description automatically generated

Problem 3:

Code:

#This code is part of:  
#  
# CMPSCI 370: Computer Vision, Spring 2021  
# University of Massachusetts, Amherst  
# Instructor: Subhransu Maji  
#  
# Mini-Project 6  
  
import pickle  
import numpy as np  
import matplotlib.pyplot as plt  
  
  
def softmax(z):  
 return 1.0/(1+np.exp(-z))  
  
def linearTrain(x, y):  
 #Training parameters  
 maxiter = 50  
 lamb = 0.01  
 eta = 0.01  
   
 #Add a bias term to the features  
 x = np.concatenate((x, np.ones((1, x.shape[1]))), axis=0)  
   
 class\_labels = np.unique(y)  
 num\_class = class\_labels.shape[0]  
 assert(num\_class == 2) # Binary labels  
 num\_feats = x.shape[0]  
 num\_data = x.shape[1]  
   
 true\_prob = np.zeros(num\_data)  
 true\_prob[y == class\_labels[0]] = 1  
   
 #Initialize weights randomly  
 model = {}  
 model['weights'] = np.random.randn(num\_feats)\*0.01  
 # print('w', model['weights'].shape)  
 #Batch gradient descent  
 verbose\_output = False  
 for it in range(maxiter):  
 prob = softmax(model['weights'].dot(x))  
 delta = true\_prob - prob  
 gradL = delta.dot(x.T)  
 model['weights'] = (1 - eta\*lamb)\*model['weights'] + eta\*gradL  
 model['classLabels'] = class\_labels  
  
 return model  
  
  
def linearPredict(model, x):  
 #Add a bias term to the features  
 x = np.concatenate((x, np.ones((1, x.shape[1]))), axis=0)  
  
 prob = softmax(model['weights'].dot(x))  
 ypred = np.ones(x.shape[1]) \* model['classLabels'][1]  
 ypred[prob > 0.5] = model['classLabels'][0]  
  
 return ypred  
  
  
def testLinear():  
 #--------------------------------------------------------------------------  
 # Your implementation to answer questions on Linear Classifier  
 #--------------------------------------------------------------------------  
 data = pickle.load(open('data\_binarized.pkl', 'rb'))  
 x, y = data['train']['x'], data['train']['y']  
  
 # Reshape from HxWx200 to Nx200, N = H\*W  
 x = np.reshape(x, (len(data['train']['x'][:, :, 1]) \* len(data['train']['x'][:, :, 1]),  
 len(data['train']['y'])))  
 model = linearTrain(x, y)  
  
 # Reshape testing data  
 x\_test = data['test']['x']  
 x\_test = np.reshape(x\_test, (len(data['test']['x'][:, :, 1]) \* len(data['test']['x'][:, :, 1]),  
 len(data['test']['y'])))  
 # Predict test data using trained model  
 ypred = linearPredict(model, x\_test)  
 acc = 0  
 for i in range(len(data['test']['y'])):  
 # Compare predict to label of testing  
 if ypred[i] == data['test']['y'][i]:  
 acc += 1  
 acc /= len(data['test']['y'])  
 print(acc\*100, '%')  
  
 # Visualization part  
 dimension = model["weights"][:len(model["weights"])-1]  
 w = np.reshape(dimension, (int(np.sqrt(len(dimension))), int(np.sqrt(len(dimension)))))  
 wp = np.clip(w, 0, None)  
 wn = np.clip(w, None, 0)  
 plt.subplot(131)  
 plt.title('Positive Weights')  
 plt.imshow(wp, cmap='gray')  
 plt.subplot(133)  
 plt.title('Negative Weights')  
 plt.imshow(wn, cmap='gray')  
 plt.show()  
  
 return None  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 testLinear()

Result:

Accuracy= around 98%

A picture containing timeline

Description automatically generated