

Applied Machine Learning: Homework #1

Due on DUE DATE: OCT 01 11:59 PM

CLASS INSTRUCTOR: Volodymyr Kuleshov

Wentao Xu(wx225) Xingyue Dai(xd86)

PROGRAMMING EXERCISES

Problem 1

Digit Recognizer

- (a) Join the Digit Recognizer competition on Kaggle. Download the training and test data. The competition page describes how these files are formatted.
- (b) Write a function to display an MNIST digit. Display one of each digit.
- (c) Examine the prior probability of the classes in the training data. Is it uniform across the digits? Display a normalized histogram of digit counts. Is it even?
- (d) Pick one example of each digit from your training data. Then, for each sample digit, compute and show the best match (nearest neighbor) between your chosen sample and the rest of the training data. Use L2 distance between the two images' pixel values as the metric. This probably won't be perfect, so add an asterisk next to the erroneous examples (if any).
- (e) Consider the case of binary comparison between the digits 0 and 1. Ignoring all the other digits, compute the pairwise distances for all genuine matches and all impostor matches, again using the L2 norm. Plot histograms of the genuine and impostor distances on the same set of axes.
- (f) Generate an ROC curve from the above sets of distances. What is the equal error rate? What is the error rate of a classifier that simply guesses randomly?
- (g) Implement a K-NN classifier. (You cannot use external libraries for this question; it should be your own implementation.)
- (h) Take 15% of your dataset and make it your holdout set. Train the classifier on the remaining data, using different values of K and use the holdout set to determine the best value of K. Print a table showing the values of K you tried as well as their training and holdout accuracy. Report the best value of K that you found.
- (i) Generate a confusion matrix (of size 10×10) from your results. Which digits are particularly tricky to classify?
- (j) Train your classifier with all of the training data, and test your classifier with the test data. Submit your results to Kaggle.

Solution

- (a) Done.
- (b) The below are codes which can show the digits.

```
import os
import numpy as np
import csv
import matplotlib.pyplot as plt
import matplotlib

def read_mnist(path):
    '''
```

```

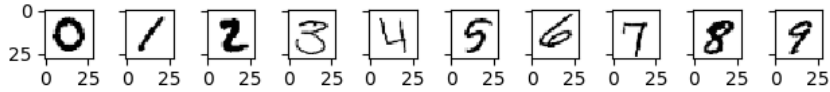
    read_mnist_dataset
    '''
    file = open(path, "r")
    reader = csv.reader(file)
    data = []
    for line in reader:
        if reader.line_num == 1:
            continue
        data.append(line)
    return data

def show_each_digit(data):
    script = []
    digit_set = set()
    matplotlib.use('TkAgg')
    figure, axis = plt.subplots(nrows=1, ncols=10, sharex='all', sharey='all')
    for i in range(0, len(data)):
        script.append([])
        for j in range(1, len(data[i]), 1):
            script[i].append(float(data[i][j]) / 255.0)
            # reshape data
        script[i] = np.array(script[i]).reshape((28, 28))
        if data[i][0] not in digit_set:
            axis[int(data[i][0])].imshow(script[i], cmap='Greys', interpolation='nearest')
            digit_set.add(data[i][0])
        else:
            continue
        if (len(digit_set) == 10):
            break
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    path = "./dataset/digit-recognizer/train.csv"
    data = read_mnist(path)
    show_each_digit(data)

```

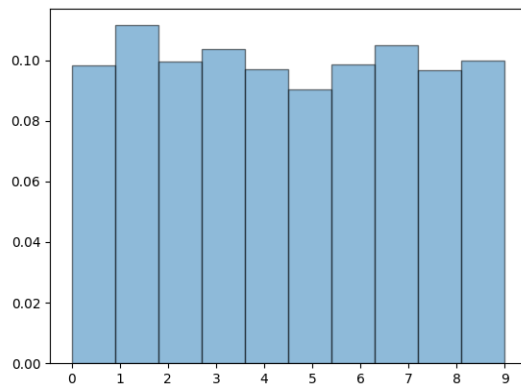
The picture below is the picture of each digits.



(c)

```
def problem_1_c(data):
    label = []
    labels_stat = [0,0,0,0,0,0,0,0,0,0]
    for i in range(len(data)):
        label.append(data[i][0])
        labels_stat[int(data[i][0])] += 1
    label = np.sort(np.array(label))
    # print(label)
    plt.hist(label, weights=np.ones(len(label)) / len(label), edgecolor='k', alpha=0.5)
    plt.savefig("./histogram.png")
```

I examined the prior probability of the classes in the training data. It is uniform and even. The histogram is shown below.



(d) The codes:

```
def problem_1_d(data):
    sample = []
```

```

sample_sign = []
sample_rank = []
digit_set = set()
# current = []
# pick sample from data set
for i in range(0, len(data)):
    current = []
    for j in range(1, len(data[i]), 1):
        current.append(float(data[i][j]))
        # reshape data
    label = int(data[i][0])
    if label not in digit_set:
        sample.append(current)
        sample_sign.append(label)
        sample_rank.append(i)
        digit_set.add(label)
    else:
        continue
    if (len(digit_set) == 10):
        break
nearest_data = [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]
nearest_label = [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]
nearest_attribute = [[],[],[],[],[],[],[],[],[],[]]
for j in range(len(sample)):
    nearest_distance = float('inf')
    for i in range(len(data)):
        data_attribute = []
        if (i == sample_rank[j]):
            continue
        for k in range(1, len(data[i]), 1):
            data_attribute.append(float(data[i][k]))
        data_label = int(data[i][0])
        sample_target = np.array(sample[j])
        data_attribute = np.array(data_attribute)
        # L2 distance
        distance = np.linalg.norm(sample_target-data_attribute)
        if distance<=nearest_distance:
            nearest_data[j] = i
            nearest_label[j] = data_label
            nearest_attribute[j] = data_attribute
            nearest_distance = distance

print("Sample label:")
print(sample_sign)
print("Nearest Neighbor label:")
print(nearest_label)
print("Sample number in data set:")
print(sample_rank)
print("Nearest number in data set:")
print(nearest_data)

```

The codes shown above is nearest neighbor method. I also printed the result of using this method:

Sample label:

[1, 0, 4, 7, 3, 5, 8, 9, 2, 6]

Nearest Neighbor label:

[1, 0, 4, 7, 5, 5, 8, 9, 2, 6]

Sample number in data set:

[0, 1, 3, 6, 7, 8, 10, 11, 16, 21]

Nearest number in data set:

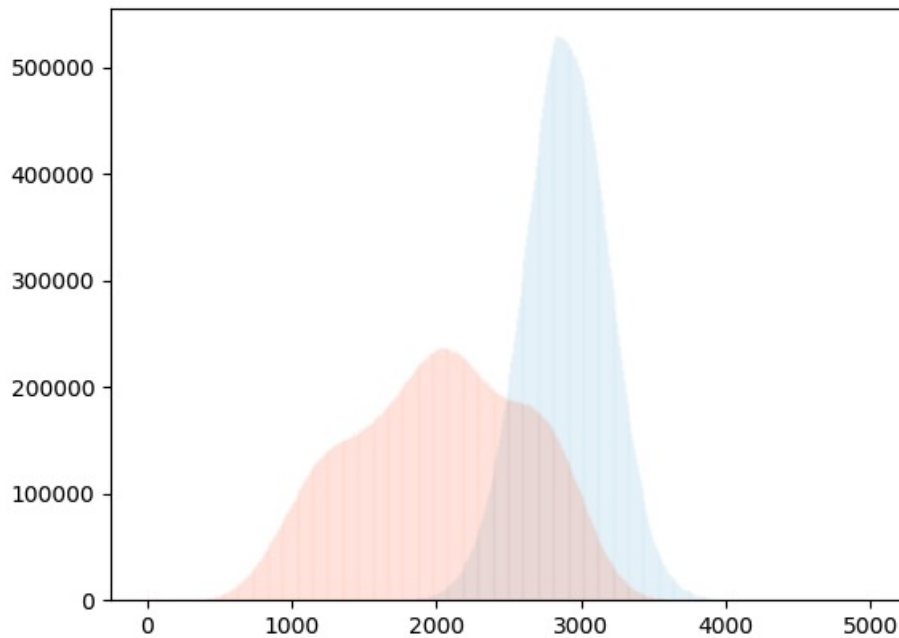
[29704, 12950, 14787, 15275, 8981, 30073, 32586, 35742, 9536, 16240]

The above is the result. Sample label means the label of samples I pick from every digit. The nearest neighbor label means the label nearest label of data picked from the method of "nearest neighbor". Here we can see the label '5' is not equal to the label '3'. I then check the original attributes based on the number I collected and found the #'8981' data is nearest to the #'7' data. So the error was generated.

```
(e) def problem_1_e(data):
    digits_1 = []
    digits_0 = []
    for i in range(len(data)):
        if data[i][0] == '0':
            digits_0.append(data[i][1:])
        if data[i][0] == '1':
            digits_1.append(data[i][1:])
        else:
            continue
    digits_0 = process_data(digits_0)
    digits_1 = process_data(digits_1)
    digits_0 = np.array(digits_0)
    digits_1 = np.array(digits_1)

    impostor_distance = []
    genuine_distance = []
    distance_0 = distance.cdist(digits_0, digits_0, 'euclidean')
    distance_1 = distance.cdist(digits_1, digits_1, 'euclidean')
    distance_01 = distance.cdist(digits_0, digits_1, 'euclidean')
    distance_0 = np.reshape(distance_0, (1,-1)).tolist()[0]
    distance_1 = np.reshape(distance_1, (1,-1)).tolist()[0]
    distance_01 = np.reshape(distance_01, (1,-1)).tolist()[0]
    genuine_distance = distance_0 + distance_1
    impostor_distance = distance_01 + distance_01
    bins = range(0, 5000, 10)
    plt.hist(impostor_distance, bins=bins, color = 'skyblue', edgecolor='skyblue', alpha=0.05)
    plt.hist(genuine_distance, bins=bins, color = 'coral', edgecolor='coral', alpha=0.05)
    plt.savefig("./one_zero.png")
```

The red hists represent genuine distance, and the blue hists represent impostor distance.



- (f) For this problem, I choose a distance as a criteria, genuine distances which are smaller than the criteria are regarded as True positive, impostor distances which are smaller than the criteria are regarded as False positive.

And then I increase distance to generate different points on ROC.

```
def problem_1_f(data):
    digits_1 = []
    digits_0 = []
    for i in range(len(data)):
        if data[i][0] == '0':
            digits_0.append(data[i][1:])
        if data[i][0] == '1':
            digits_1.append(data[i][1:])
        else:
            continue
    digits_0 = process_data(digits_0)
    digits_1 = process_data(digits_1)
    digits_0 = np.array(digits_0)
    digits_1 = np.array(digits_1)

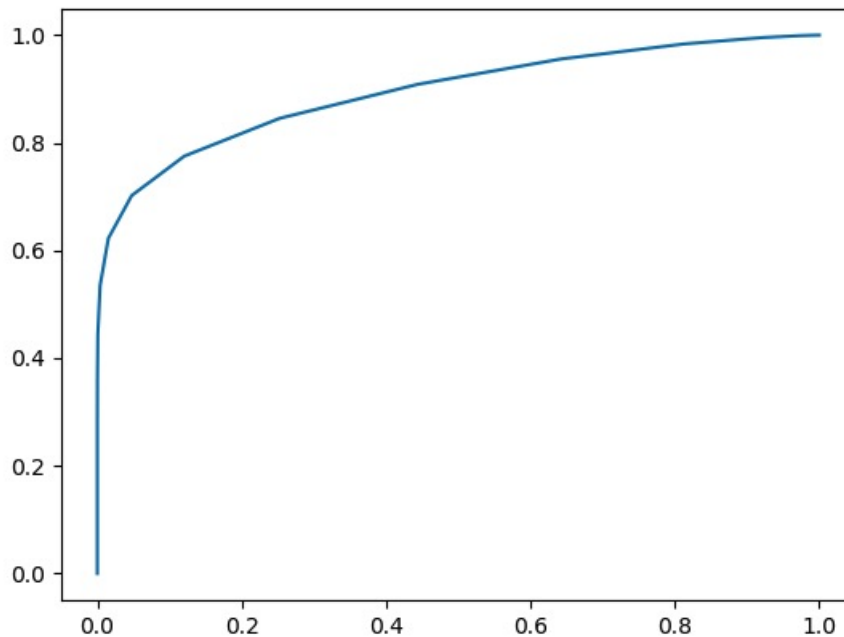
    impostor_distance = []
    genuine_distance = []
    distance_0 = distance.cdist(digits_0, digits_0, 'euclidean')
    distance_1 = distance.cdist(digits_1, digits_1, 'euclidean')
    distance_01 = distance.cdist(digits_0, digits_1, 'euclidean')
    distance_0 = np.reshape(distance_0, (1,-1)).tolist()[0]
    distance_1 = np.reshape(distance_1, (1,-1)).tolist()[0]
    distance_01 = np.reshape(distance_01, (1,-1)).tolist()[0]
```

```

genuine_distance = distance_0 + distance_1
impostor_distance = distance_01 + distance_01
roc_distance = 0
max_distance = 0
for i in range(len(genuine_distance)):
    if genuine_distance[i] > max_distance:
        max_distance = genuine_distance[i]
TPR = []
FPR = []
for roc_distance in range(0, round(max_distance), 150):
    tpr = 0
    fpr = 0
    print("process:{0}%".format(round((roc_distance) * 100 / max_distance)), end="\r")
    for t in range(len(genuine_distance)):
        if genuine_distance[t] < roc_distance:
            tpr += 1
    for f in range(len(impostor_distance)):
        if impostor_distance[f] < roc_distance:
            fpr += 1
    tpr = tpr / len(genuine_distance)
    fpr = fpr / len(impostor_distance)
    TPR.append(tpr)
    FPR.append(fpr)
plt.plot(FPR, TPR)
plt.savefig("./ROC.png")

```

The below is ROC.



Based on the definition of EER, the Equal Error rate should be the common value when the false

acceptance rate and false rejection rate are equal, so the EER is approximately 0.27.

I don't really understand the "error rate of a classifier that simply guesses randomly.", I think if the classifier guesses randomly, the error rate is approximately 0.5, since the numbers of digit 0 and digit 1 are approximately the same.

- (g) I implemented kNN using three methods, simple knNN, kdtree kNN and cdist KNN. Here I put kdtree knn here, the rest will be submitted in .zip file.

```
class Node:
    def __init__(self, data, split=0, left=None, right=None):
        self.data = data
        self.split = split
        self.left = left
        self.right = right

class KDTree:
    def create(self, data, split):
        if (len(data) == 0):
            return None
        dim = len(data[0])
        data = sorted(data, key=lambda x: x[split])
        pivotal = int(len(data) / 2)
        mid_point = data[pivotal]
        root = Node(mid_point, split)
        root.left = self.create(data[: pivotal], (split+1)%dim)
        root.right = self.create(data[pivotal+1:], (split+1)%dim)
        return root

    def __init__(self, data):
        self.root = self.create(data=data, split=0)

    def KNN(self, data_point, dic, k=1):
        k_nearest_data = []
        for i in range(k):
            k_nearest_data.append([-1, None])
        self.k_nearest_data = np.array(k_nearest_data)

    def search(point, node, split, dic):
        if (node != None):
            d = point[split] - node.data[split]
            dim = len(point)
            if d < 0:
                search(point, node.left, (split+1)%dim, dic)
            else:
                search(point, node.right, (split+1)%dim, dic)
            distance = np.linalg.norm(point - node.data)
            for i, dd in enumerate(self.k_nearest_data):
                if (dd[0] < 0 or dd[0] > distance):
                    self.k_nearest_data = np.insert(self.k_nearest_data, i, [distance,
                    self.k_nearest_data = self.k_nearest_data[:-1]
```

```

        break
    # compare the other side of KDTree
    faarest = list(self.k_nearest_data[:, 0]).count(-1)
    # there may exist data points in other side which is closer to the target
    if self.k_nearest_data[-faarest-1, 0] > abs(d):
        if d >= 0:
            search(point, node.left, (split+1)%dim, dic)
        else:
            search(point, node.right, (split+1)%dim, dic)

split=0
search(data_point, self.root, split, dic)
label_stat = [0,0,0,0,0,0,0,0,0,0]
for i in range(len(self.k_nearest_data)):
    label_stat[int(self.k_nearest_data[i][1])] += 1
result = -1
answer = -1
for i in range(len(label_stat)):
    if label_stat[i] > result:
        result = label_stat[i]
        answer = i
return answer

```

- (h) I get holdout accuracy for k=2,3,4,5,6,7,8, the training accuracy took me much time, so I just ran k = 3,4,5,6 to get training accuracy for different k. My configuration:

```

Architecture:      x86_64
CPU op-mode(s):    32-bit, 64-bit
Byte Order:        Little Endian
CPU(s):            32
On-line CPU(s) list: 0-31
Thread(s) per core: 2
Core(s) per socket: 16
Socket(s):         1
NUMA node(s):      1
Vendor ID:         GenuineIntel
CPU family:        6
Model:             79
Model name:        Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz
Stepping:          1
CPU MHz:           2261.297
CPU max MHz:       3000.0000
CPU min MHz:       1200.0000
BogoMIPS:          4600.03
Hypervisor vendor: Xen
Virtualization type: full
L1d cache:         32K
L1i cache:         32K
L2 cache:          256K
L3 cache:          46080K
NUMA node0 CPU(s): 0-31
Flags:             fpu vme de pse tsc msr pae mce cx8 apic sep mtrr codes:

```

```

def split_data(data, p=0.15):
    split = []
    max_len = int(p * len(data))
    while(len(split) < max_len):

```

```

        index = int(random.random() * len(data))
        split.append(data[index])
        temp = data[index]
        data.remove(temp)
    return split, data

def simple_split(data, p=0.15):
    split = []
    max_len = int(p * len(data))
    split = data[0:max_len+1]
    data = data[max_len+1:]
    return split, data

def get_label(data):
    label = []
    for i in range(len(data)):
        label.append(int(data[i][0]))
    return label

def problem_1_h(data, KK):
    split, data = split_data(data, 0.15)
    data = process_data(data)
    dic, dat = data_dic(data)
    # build KDTTree
    K = KDTree(np.array(dat))
    split_dic, split_dat = data_dic(split)
    # split data
    K_holdout_list = []
    K_train_list = []
    count = 0
    for i in range(len(KK)):
        accuracy = 0
        train_acc = 0
        # training accuracy
        # for t in range(len(data)):
        #     print("process:{0}%".format(round((count) * 100 / (len(KK)*42000))), end="\r")
        #     count += 1
        #     predict_label = K.KNN(np.array(dat[t]), dic, KK[i])
        #     # print(predict_label == int(data[t][0]))
        #     if (predict_label == int(data[t][0])):
        #         train_acc += 1
        # train_acc = float(train_acc) / float(len(data))
        # K_train_list.append(train_acc)

        # holdout accuracy
        for j in range(len(split)):
            print("process:{0}%".format(round((count) * 100 / (len(KK)*6300))), end="\r")
            count += 1

```

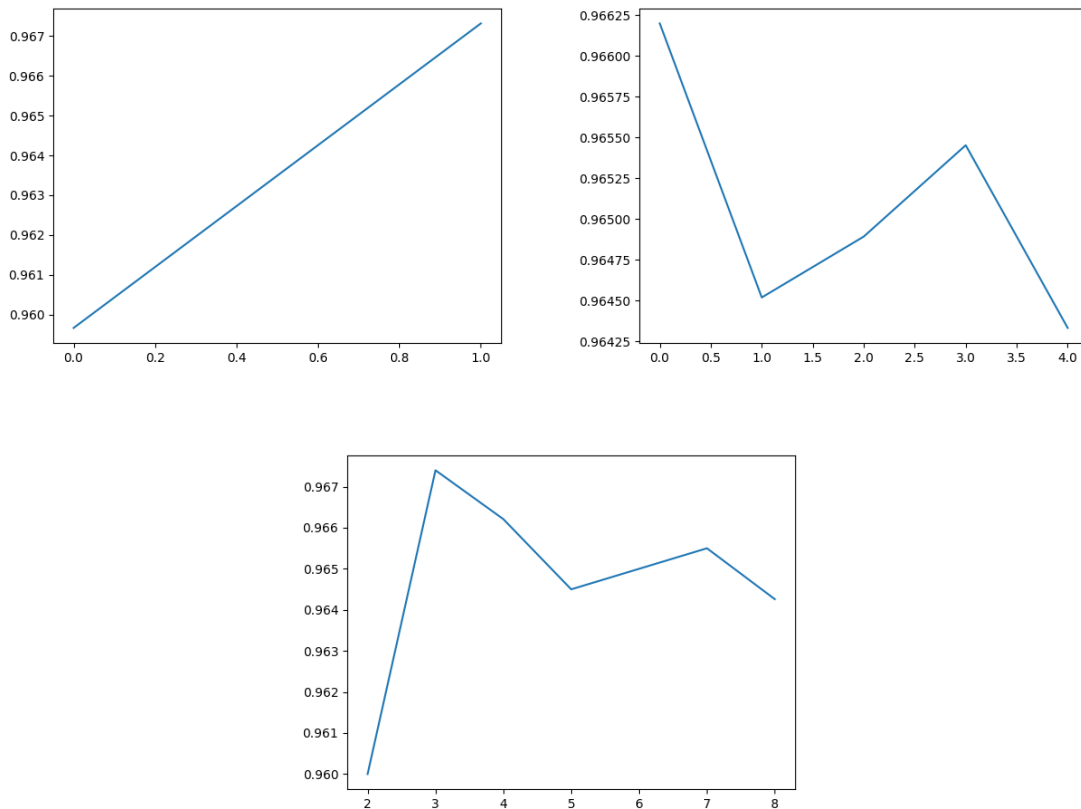
```

    # predict_label = simple_KNN(split[j], data, K[i])
    predict_label = K.KNN(np.array(split_dat[j]), dic, KK[i])
    # print(predict_label == int(split[j][0]))
    if (predict_label == int(split[j][0])):
        accuracy += 1
    accuracy = float(accuracy) / float(len(split))
    K_holdout_list.append(accuracy)
# print(K_train_list)
print(K_holdout_list)
plt.figure()
# plt.plot(range(len(K_train_list)), K_train_list)
# plt.savefig("./KNN_train.png")

plt.figure()
plt.plot(range(len(K_holdout_list)), K_holdout_list)
plt.savefig("./KNN_hold_out.png")

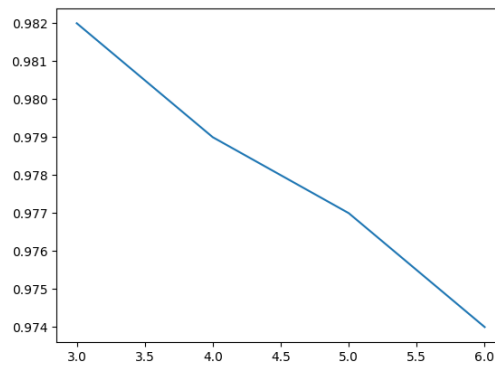
```

Here since it took me too much time, I run it twice and merge. The below are two graphs of holdout accuracy, ran $k = 2, 3$ and $k = 4, 5, 6, 7$



The below is merge one:

The x-coord represents k .



The below is training accuracy.

So I choose $k = 3$.

(i) I run holdout set to get confusion matrix. Below are codes:

```
def problem_1_i(data, KK):
    split, data = simple_split(data, 0.15)
    data = process_data(data)
    split = process_data(split)
    data_lab = get_label(data)
    dic, dat = data_dic(data)
    split_dic, split_dat = data_dic(split)
    total_label = []
    train_label = []

    for i in range(len(split)):
        total_label.append(data_lab)
    for i in range(len(dat)):
        train_label.append(data_lab)

    K_h = []
    K_t = []
    count = 0
    y_true = []
    y_pred = []
    for i in range(len(KK)):
        # holdout data
        distance = matrix_distance.cdist(split_dat, dat, 'euclidean')
        index = np.argsort(np.array(distance)[0:,])
        index = np.split(index, (KK[i],), axis=1)[0]
        total_label = np.array(total_label)
        # label = []
        holdout_predict = []
        for j in range(len(total_label)):
            print("process:{0}%".format(round((count) * 100 / (6300 * len(KK)))), end="\r")
            count += 1
            temp_label = ((np.array(total_label)[j])[index[j]]).tolist()
            label_stat = [0,0,0,0,0,0,0,0,0,0]
```

```

# temp_label = label[j]
for t in range(len(temp_label)):
    current_t = int(temp_label[t])
    label_stat[current_t] += 1
result = -1
answer = -1
# print(label_stat)
for t in range(len(label_stat)):
    if label_stat[t] > result:
        result = label_stat[t]
        answer = t
y_pred.append(answer)
y_true.append(int(split[j][0]))
if (answer == int(split[j][0])):
    # print(True)
    holdout_predict.append(1)
else:
    # print(False)
    holdout_predict.append(0)
hold_acc = np.array(holdout_predict).sum() / float(len(holdout_predict))
K_h.append(hold_acc)
print(confusion_matrix(y_true, y_pred))

```

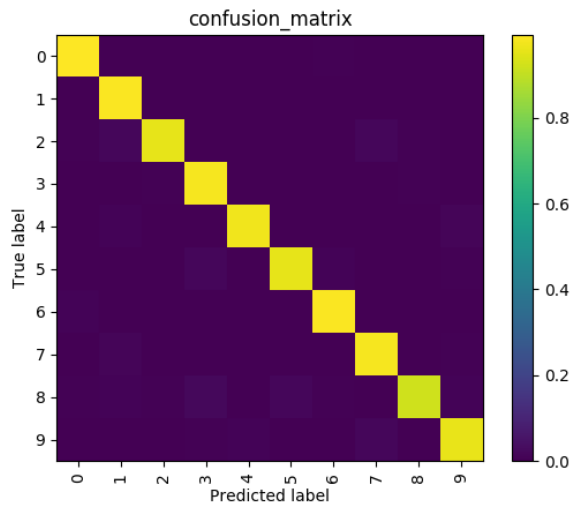
The confusion matrix:

```

[[621 0 0 0 0 0 3 1 0 0]
 [ 0 676 1 1 1 0 2 1 1 1]
 [ 3 12 662 1 0 0 0 11 3 0]
 [ 1 1 3 597 0 1 0 2 4 1]
 [ 0 5 0 0 585 0 1 2 0 9]
 [ 0 2 1 9 0 543 6 1 1 4]
 [ 5 1 0 0 1 1635 0 0 0 0]
 [ 0 9 0 0 1 0 0 644 0 4]
 [ 4 7 3 14 0 10 4 1554 5]
 [ 1 1 0 3 7 0 2 10 1593]]

```

And I used matplotlib to plot it. The below:



So, from the matrix, we can see that digit 8 is tricky to classify.

(j) codes:

```
def problem_1_j(data, test_data):
    data = process_data(data)
    dic, dat = data_dic(data)
    # build KDTree
    K = KDTree(np.array(dat))
    test_data = process_data(test_data)
    answer = []
    id = []
    for i in range(len(test_data)):
        print("process:{0}%".format(round((i) * 100 / len(test_data))), end="\r")
        id.append(i+1)
        answer.append(K.KNN(np.array(test_data[i]), dic, 3))
    ex = {"ImageId":id,
          "Label":answer}
    d=DataFrame(ex)
    d.to_csv("./result.csv", index=False)

if __name__ == "__main__":
    path = "./dataset/digit-recognizer/train.csv"
    data = read_mnist(path)
    test = "./dataset/digit-recognizer/test.csv"
    test_data = read_mnist(test)
    # print(test_data)
    problem_1_j(data, test_data)
```

The score:

2101	WentXu		0.96582	1	1d
Your First Entry 					
Welcome to the leaderboard!					

Problem 2

Predicting HousePrices

- (a) Join the House Prices competition on Kaggle. Download the training and test data.
- (b) Using least squares, try to predict house prices on this dataset. You can choose the features (or combinations of features) you would like to use or ignore, provided you justify your reasoning.
- (c) Now try predicting house prices using regularized least squares. Try both L1 and L2 regularization and compare them. Briefly describe your findings from training regularized and unregularized models.
- (d) Train your model using all of the training data, and test it using the testing data. Submit your results to Kaggle.

Solution

- (a) Done.
- (b) I tried to implement the mse method and gradient methods, but it doesn't converge, so I turned to use sklearn.

Below are my own implementation, but it doesn't converge:

```
def f(X, theta):
    y_prime = X.dot(theta)
    return y_prime

def lsq_gradient(theta, X, y):
    a = 2 * (X.T.dot(X)).dot(theta) - 2 * X.T.dot(y)
    return a

def least_square(theta, X, y):
    y_prime = f(X, theta)
    return np.linalg.norm(y_prime - y)

def train(X_train, y_train, threshold=1e-3, step_size=4e-1):
    threshold = 1e-3
    step_size = 4e-1
    theta, theta_prev = np.zeros(shape=(X_train.shape[1],1)), (np.zeros(shape=(X_train.shape[1],1)))
    opt_pts = [theta]
    opt_grads = []
    iter = 0
    while np.linalg.norm(theta - theta_prev) > threshold:
        if iter % 100 == 0:
            print('Iteration %d. LSQ: %.6f' % (iter, least_square(theta, X_train, y_train)))
        theta_prev = theta
        gradient = lsq_gradient(theta, X_train, y_train)
        theta = theta_prev - step_size * gradient
        opt_pts += [theta]
        opt_grads += [gradient]
        iter += 1
```


I choose 2, 4, 5, 18, 19, 20, 21, 27, 35, 37, 38, 39, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 55, 57, 60, 62, 63, 68, 69, 70, 71, 72, 76, 77, 78th attributes. Since these attributes are numerical, other attributes consist of different name or signs, I think discrete attributes involves one-hot or embedding vector process method, so here I just deleted it.

Below are my data process codes, I replace 'NA' with the average value of that column:

```
def process(data, col):
    a = []
    data = np.array(data)
    for i in range(len(data)):
        a.append(data[i][col].tolist())
    attri_avg = []
    for j in range(len(a[0])):
        temp = 0
        count = 0
        for i in range(len(a)):
            if a[i][j] != 'NA':
                temp = temp + float(a[i][j])
                count += 1
            else:
                continue
        temp = temp / count
        attri_avg.append(temp)

    for i in range(len(a)):
        for j in range(len(a[i])):
            if (a[i][j] == 'NA'):
                a[i][j] = attri_avg[j]
            else:
                a[i][j] = float(a[i][j])
    return a
```

I used LinearRegression in sklearn and the score is 0.8119459869658459.

(c) I just used Ridge and Lasso in sklearn. Codes:

```
clf = Ridge(alpha=1.0)
clf.fit(X, Y)
print(clf.score(X, Y))
lasso = Lasso(alpha=1.0)
lasso.fit(X, Y)
print(lasso.score(X, Y))
```



and the scores:

0.8119453413954714

0.8119459699679068

And for both L1 and L2 regularized least squares method, when I increase alpha, the score of both models decrease, so I think that it is because all the attributes I chose are useful in this predicting problem.

- (d) Since I found when alpha becomes larger and larger, the score of classifier becomes lower and lower, so I just use linearRegression to predict the test dataset. Below is the score on kaggle:

4641	WentXu		0.25091	1	20h
Your First Entry 					
Welcome to the leaderboard!					

WRITTEN EXERCISES

Problem 3

Maximum Likelihood and KL Divergence. Let $\hat{p}(x, y)$ denote the empirical data distribution over a space of inputs $x \in \mathcal{X}$ and outputs $y \in \mathcal{Y}$. For example, in an image recognition task, x can be an image and y can be whether the image contains a cat or not. Let $p_\theta(y|x)$ be a probabilistic classifier parameterized by θ , e.g., a logistic regression classifier with coefficients θ . Show that the following equivalence holds:

$$\arg \max_{\theta \in \Theta} \mathbb{E}_{\hat{p}(x, y)} [\log p_\theta(y|x)] = \arg \min_{\theta \in \Theta} \mathbb{E}_{\hat{p}(x)} [KL(\hat{p}(y|x) || p_\theta(y|x))]$$

where KL denotes the KL-divergence:

$$KL(p(x) || q(x)) = \mathbb{E}_{x \sim p(x)} [\log p(x) - \log q(x)]$$

Solution First we can take a look at the right side of equation:

$$\begin{aligned} \arg \min_{\theta \in \Theta} \mathbb{E}_{\hat{p}(x)} [KL(\hat{p}(y|x) || p_\theta(y|x))] &= \arg \min_{\theta \in \Theta} \mathbb{E}_{\hat{p}(x)} [\mathbb{E}_{\hat{p}(y|x)} \log \hat{y}(y|x) - \mathbb{E}_{\hat{p}(y|x)} \log p_\theta(y|x)] \\ &= \arg \min_{\theta \in \Theta} \mathbb{E}_{\hat{p}(x)} [\mathbb{E}_{\hat{p}(y|x)} \log \hat{y}(y|x)] - \arg \min_{\theta \in \Theta} \mathbb{E}_{\hat{p}(y|x)} \log p_\theta(y|x) \end{aligned} \quad (1)$$

Since the first part of argmin is fixed, so we can delete it

$$\begin{aligned} \arg \min_{\theta \in \Theta} \mathbb{E}_{\hat{p}(x)} [KL(\hat{p}(y|x) || p_\theta(y|x))] &= \arg \min_{\theta \in \Theta} \mathbb{E}_{\hat{p}(x)} [\mathbb{E}_{\hat{p}(y|x)} \log \hat{y}(y|x) - \mathbb{E}_{\hat{p}(y|x)} \log p_\theta(y|x)] \\ &= \arg \min_{\theta \in \Theta} \mathbb{E}_{\hat{p}(x)} [\mathbb{E}_{\hat{p}(y|x)} \log \hat{y}(y|x)] - \arg \min_{\theta \in \Theta} \mathbb{E}_{\hat{p}(y|x)} \log p_\theta(y|x) \\ &= - \arg \min_{\theta \in \Theta} \mathbb{E}_{\hat{p}(x)} \mathbb{E}_{\hat{p}(y|x)} \log p_\theta(y|x) \\ &= \arg \max_{\theta \in \Theta} \mathbb{E}_{\hat{p}(x)} \mathbb{E}_{\hat{p}(y|x)} \log p_\theta(y|x) \end{aligned} \quad (2)$$

Here we can take a look at $\mathbb{E}_{\hat{p}(x)} \mathbb{E}_{\hat{p}(y|x)} \log p_\theta(y|x)$

$$\begin{aligned} \mathbb{E}_{\hat{p}(x)} \mathbb{E}_{\hat{p}(y|x)} \log p_\theta(y|x) &= \sum_x \sum_y \hat{p}(x) \hat{p}(y|x) \log p_\theta(y|x) \\ &= \sum_{x, y} \hat{p}(x, y) \log p_\theta(y|x) \end{aligned} \quad (3)$$

Here the derivative of (3) is based on conditional likelihood: $p(x, y) = p(y|x)p(x)$

$$\begin{aligned} \sum_{x, y} \hat{p}(x, y) \log p_\theta(y|x) &= \sum_x p(x) \sum_y \hat{p}(y|x) \log p_\theta(y|x) \\ &= \sum_x \sum_y \hat{p}(x) \hat{p}(y|x) \log p_\theta(y|x) \end{aligned} \quad (4)$$

Now we can see that based on the definition of Expectation, equation 3 can be

$$\begin{aligned}
\mathbb{E}_{\hat{p}(x)}\mathbb{E}_{\hat{p}(y|x)} \log p_{\theta}(y|x) &= \sum_x \sum_y \hat{p}(x)\hat{p}(y|x) \log p_{\theta}(y|x) \\
&= \sum_{x,y} \hat{p}(x,y) \log p_{\theta}(y|x) \\
&= \mathbb{E}_{\hat{p}(x,y)} \log p_{\theta}(y|x)
\end{aligned} \tag{5}$$

So the right side of the equation we need to prove becomes

$$\arg \max_{\theta \in \Theta} \mathbb{E}_{\hat{p}(x,y)} \log p_{\theta}(y|x)$$

which is the same as the left side of equation we need to prove.

Problem 4

Bayes rule for quality control. You're the foreman at a factory making ten million widgets per year. As a quality control step before shipment, you create a detector that tests for defective widgets before sending them to customers. The test is uniformly 95% accurate, meaning that the probability of testing positive given that the widget is defective is 0.95, as is the probability of testing negative given that the widget is not defective. Further, only one in 100,000 widgets is actually defective.

- (a) Suppose the test shows that a widget is defective. What are the chances that it's actually defective given the test result?
- (b) If we throw out all widgets that are test defective, how many good widgets are thrown away per year? How many bad widgets are still shipped to customers each year?

Solution

- (a) First we define not defective as F, defective as T. Here we construct a probability distribution:

$$p(y, \hat{y})$$

Here \hat{y} represents the result of test, y represents the real value. According to the problem, we have

$$p(y, \hat{y})/p(y) = 0.95$$

$$p(y, \hat{y}) = 0.95 * p(y)$$

Here \hat{y} and y are both F or both T, which means their value are the same. And we can know that $p(y) = F$ is $1/100,000$, $p(y) = T$ is $1 - 1/100,000$. So we get

$$p(y = F, \hat{y} = F) = 0.95 * (1 - 1/100,000)$$

$$p(y = T, \hat{y} = T) = 0.95 * 1/100,000$$

$$p(y = T, \hat{y} = F) = 0.05 * 1/100,000$$

$$p(y = F, \hat{y} = T) = 0.05 * (1 - 1/100,000)$$

So

$$\begin{aligned} p(\hat{y} = T) &= p(y = F, \hat{y} = T) + p(y = T, \hat{y} = T) \\ &= 0.95 * 1/100,000 + 0.05 * (1 - 1/100,000) \\ &= 0.050009 \end{aligned}$$

So $p(y = T, \hat{y} = T)/P(\hat{y} = T) = (0.95 * 1/100,000)/0.050009 = 0.0001899$. So the chances that it's actually defective given the result is 0.01899%.

- (b) The defective rate of testing is

$$p(y = F, \hat{y} = T) = 0.05 * (1 - 1/100,000) = 0.0499995$$

$$\#goods = 0.0499995 * 10,000,000 = 499995$$

So, 499995 goods are thrown away.

The false positive rate is

$$p(y = T, \hat{y} = F) = 0.05 * 1/100,000$$

$$\#goods = 0.05 * 1/100,000 * 10,000,000 = 5$$

So, only 5 bad goods are still shipped.

Problem 5

In k-nearest neighbors, the classification is achieved by majority vote in the vicinity of data. Suppose our training data comprises n data points with two classes, each comprising exactly half of the training data, with some overlap between the two classes.

- (a) Describe what happens to the average 0-1 prediction error on the training data when the neighbor count k varies from n to 1. (In this case, the prediction for training data point x_i includes (x_i, y_i) as part of the example training data used by kNN.)
- (b) We randomly choose half of the data to be removed from the training data, train on the remaining half, and test on the held-out half. Predict and explain with a sketch how the average 0-1 prediction error on the held-out validation set might change when k varies? Explain your reasoning.
- (c) In kNN, once k is determined, all of the k -nearest neighbors are weighted equally in deciding the class label. This may be inappropriate when k is large. Suggest a modification to the algorithm that avoids this caveat.

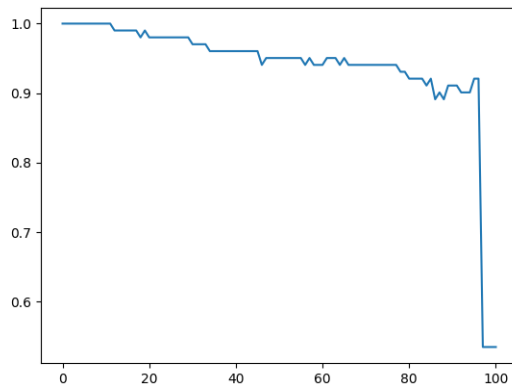
Solution

- (a) If k is n , then the average 0-1 prediction error approximately equal to 0.5. Since when k is n , the numbers of "0" data points and "1" data points are almost the same, the kNN algorithm will return every result of almost $n/2$ "0" data points and $n/2$ "1" data points, which means the result of kNN algorithm is almost the same as random guessing.

When k varies from n to 1, the prediction error becomes smaller and smaller, and it will reach a smallest value where k is very close to 1, in this case, the k is the optimal k of kNN algorithm for this problem.

- (b) I used the codes from problem 1 in code part to check the result. I picked 100 data from MNIST which only contains digit 0 and digit 1, and I split 50% as holdout and do the test.

Below are the graph:



The x-coord represents K value, and the y-coord represents holdout accuracy. So the graph shows that when K varies from n to 1, the prediction becomes larger and larger. When $k = n$, the prediction accuracy is approximately 0.5, when it becomes close to 1, the prediction accuracy becomes closer and closer to 1.0.

- (c) The modification method: if we can use a weighted kNN algorithm, we can avoid this caveat.

When we choose K nearest neighbors and start to vote, we can use $1/(\text{sum of distance})$ as the weight of the number of different-class neighbors:

$$Y = \underset{x \text{ class} \in \text{class of neighbors}}{\arg \max} \left(\sum 1/(\text{distance from data points in } x \text{ class to current data}) * \# \text{ data points in } x \text{ class} \right)$$

The prediction result is then Y.