CS5785 Fall 2017 HW1
Applied Machine Learning

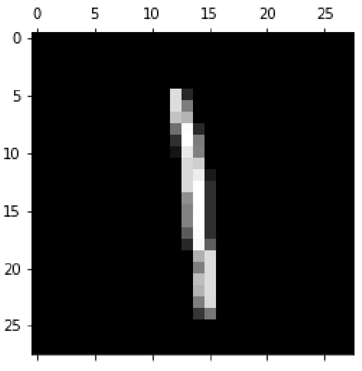
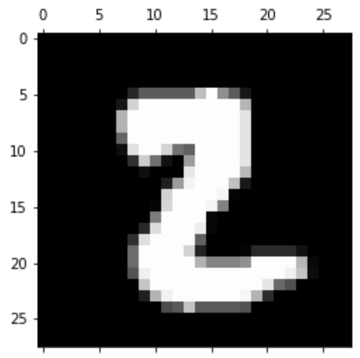
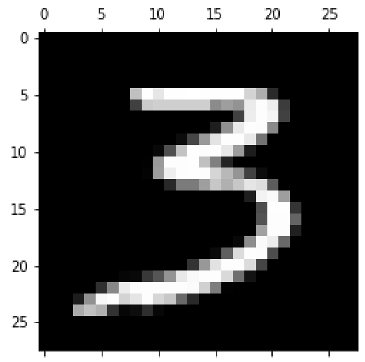
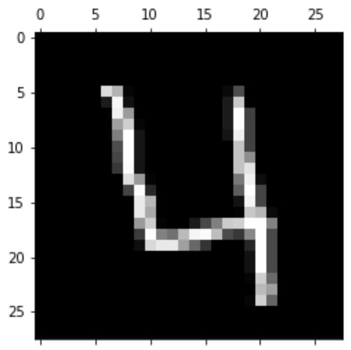
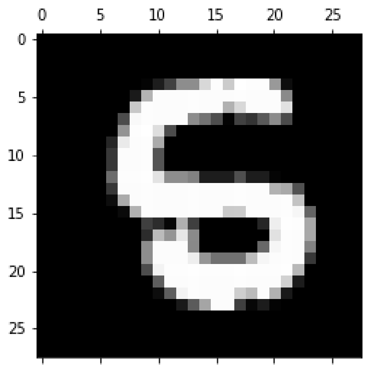
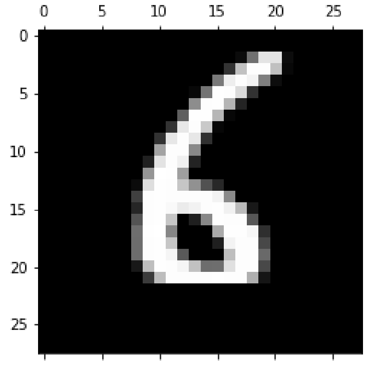
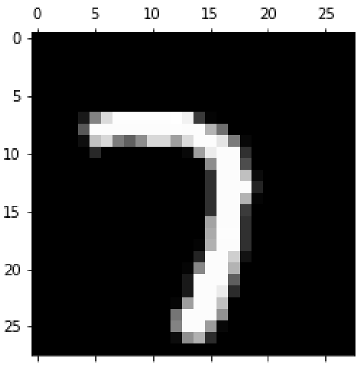
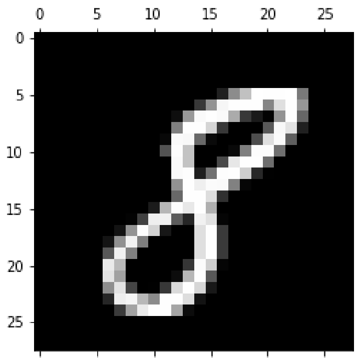
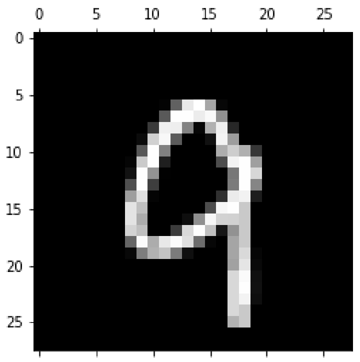
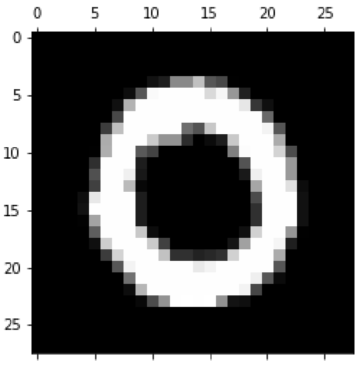
Ray Su (zs349)
Zhiyuan Jia (zj74)

Programming Exercises

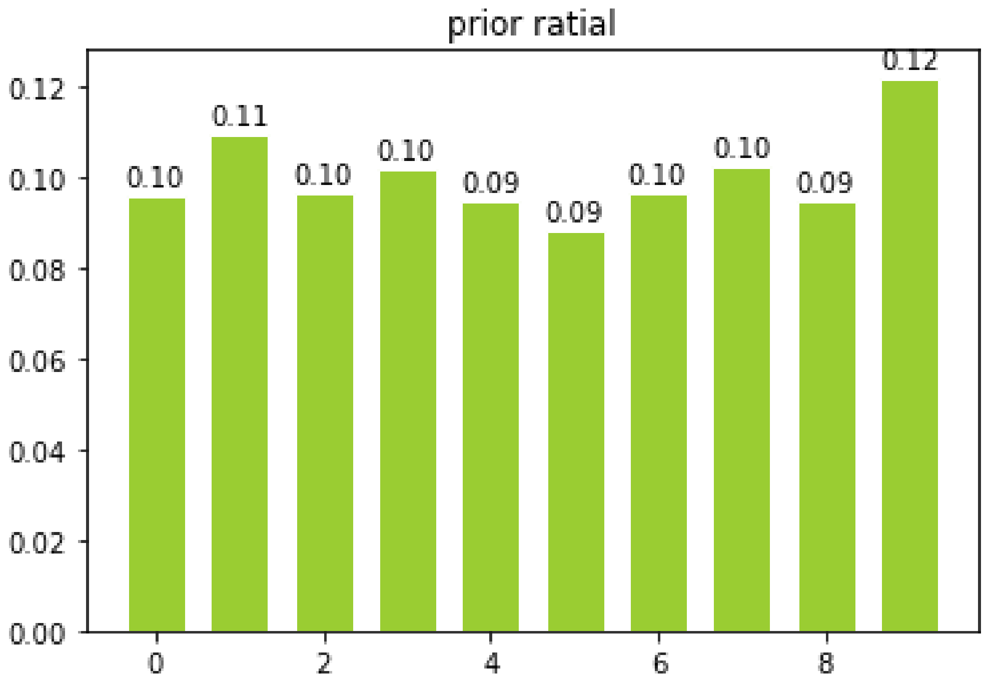
1. Digit Recognizer

(a) Please see below

(b) Write the function to display each digit

(c) Refer to the labels in the training data and count the number of each digits
Plot the data from the counter as a histogram.



The prior probability shows as follows:

[0.09838095238095237, 0.11152380952380953, 0.09945238095238096, 0.1035952380952381, 0.09695238095238096, 0.09035714285714286, 0.0985, 0.10478571428571429, 0.09673809523809523, 0.09971428571428571]

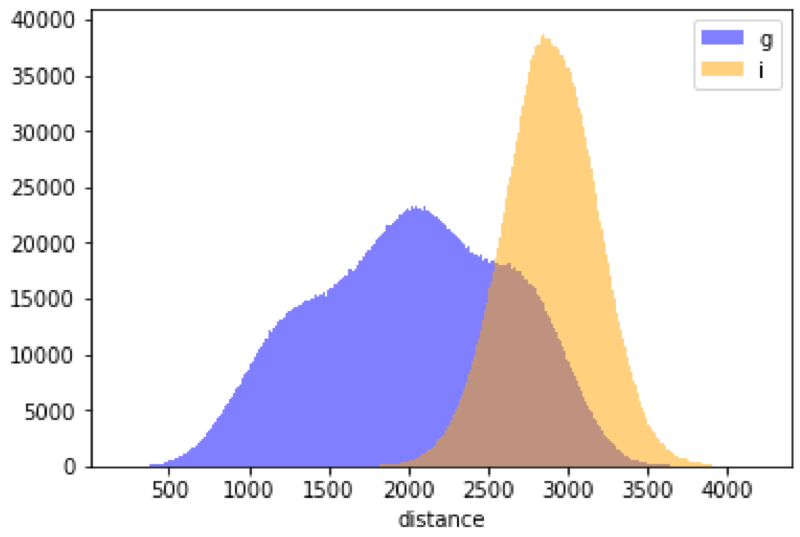
The array above represents prior probabilities from '0' to '9'
We can tell from the data that the displace of the ten digits is not quite even

(d) Just pick all these ten digits from the training data randomly. The result shows as follow. Accuracy of the nearest neighbor is at a very high level. In the process of measuring the nearest neighbor, I removed the test digit itself from the measured dataset so the distance will not be 0.

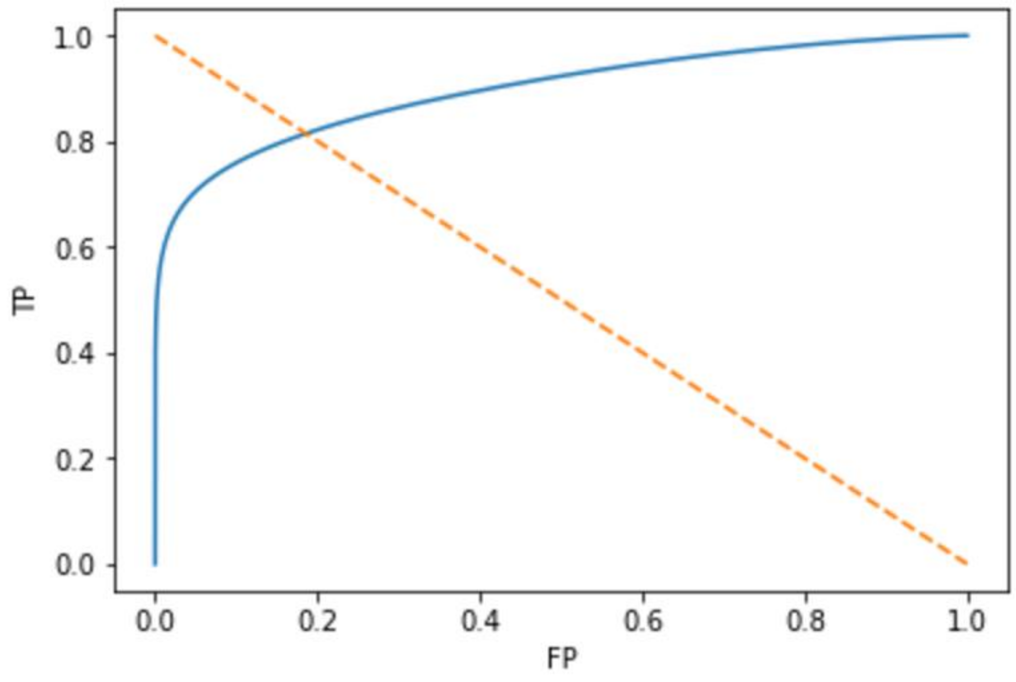
```
the No.12951 resembles sample 0 most, and the lable is 0
the No.37218 resembles sample 1 most, and the lable is 1
the No.9537 resembles sample 2 most, and the lable is 2
the No.16124 resembles sample 3 most, and the lable is 3
the No.14788 resembles sample 4 most, and the lable is 4
the No.30074 resembles sample 5 most, and the lable is 5
the No.34339 resembles sample 6 most, and the lable is 6
the No.23425 resembles sample 7 most, and the lable is 7
the No.14341 resembles sample 8 most, and the lable is 8
the No.4579 resembles sample 9 most, and the lable is 9
```

Ray Su (zs349)
Zhiyuan Jia (zj74)

(e) (1) Extract all 0s and 1s from the training dataset
(2) Pdist function to calculate the distance of genuine pairs
(3) Cdist function to compute the distance of imposter pairs
(4) Plot the histogram



(f) The ROC curve is shown below:
The x axis represents FP rate and the y axis represents TP rate
The cross point is (0.80829609657, 0.174032906816). Therefore, the error rate is 0.36573681
There are only two possibilities (0 and 1), so the guess error rate should be 0.5



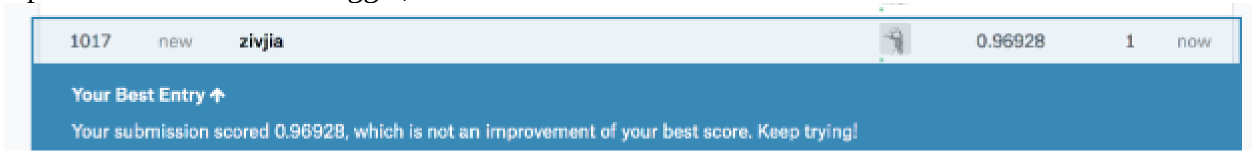
(g)
```
def knnClassifierunit(Train_data,Label,Test_data,k):                #test 1 instance
    Distance = []
    labels = []
    a = []
    Neighbors = [0 for i in range(k)]
    trainlenth = len(Train_data)

    for i in range(trainlenth):
        dist = distance.euclidean(Train_data[i],Test_data)
        Distance.append((Train_data[i],Label[i],dist))
    Distance.sort(key = itemgetter(2))
    for i in range(k):
        Neighbors[i] = (Distance[i][1])
    a = Counter(Neighbors).most_common(1)
    return a[0][0]


def knnClassifier(Train_data,Label,Test_data,k):                    #test a matrix
    Distance = []
    Neighbors = [0 for i in range(k)]
    trainlenth = len(Train_data)
    testlenth = len(Test_data)
    a = [0 for i in range(testlenth)]

    dist = distance.cdist(Test_data,Train_data)
    for i in range(testlenth):
        Distance = dist[i]
        b = sorted(range(len(Distance)),key=lambda x:Distance[x])
        for j in range(k):
            Neighbors[j] = (Label[b[j]])
        a[i] = Counter(Neighbors).most_common(1) [0] [0]
    return a
```

(h) The three-fold cross validation is used to identify a best value of K
We tested several values: [3, 5, 7, 9, 11]
And the accuracy in that order is: [[0.9662, 0.9656, 0.9638, 0.9623, 0.9603]
So, the best value for the dataset and the classifier is K = 3

(i)
```
[[4106    2    3    0    0    4   13    1    1    2]
 [   0 4658    5    3    2    1    2    8    2    3]
 [  26   35 4016   11    2    2    3   63   12    7]
 [   4    9   20 4182    0   46    4   23   42   21]
 [   4   38    0    0 3918    0   12    4    0   96]
 [  12    3    3   48    1 3640   43    2   12   31]
 [  24    7    1    1    6   14 4084    0    0    0]
 [   2   46   11    1    7    0    0 4286    0   48]
 [  12   40   11   44   13   55   14    6 3830   38]
 [  11    5    2   20   46    8    2   56   17 4021]]
```

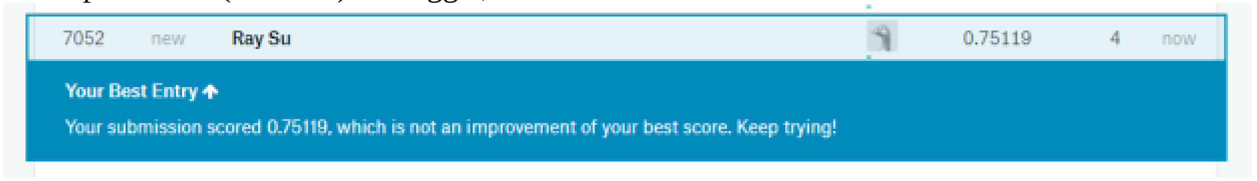
Every row represents a genuine digit was classified as the column digit
Count the wrong times for each genuine digit, we get the array:
[26, 26, 161, 169, 154, 155, 53, 115, 233, 167]
So 8s are really hard to recognize among all digits

(j) Uploaded estimation to Kaggle, with the score shown below:


1017 new zivjia 0.96928 1 now
Your Best Entry ↑
Your submission scored 0.96928, which is not an improvement of your best score. Keep trying!

2. The Titanic Disaster

(a) We have followed the instructions
(b) We conducted some feature engineering before running the logistic regression. For details, please see the attached python code and comments
(c) Uploaded prediction (attached) to Kaggle, with score shown below:

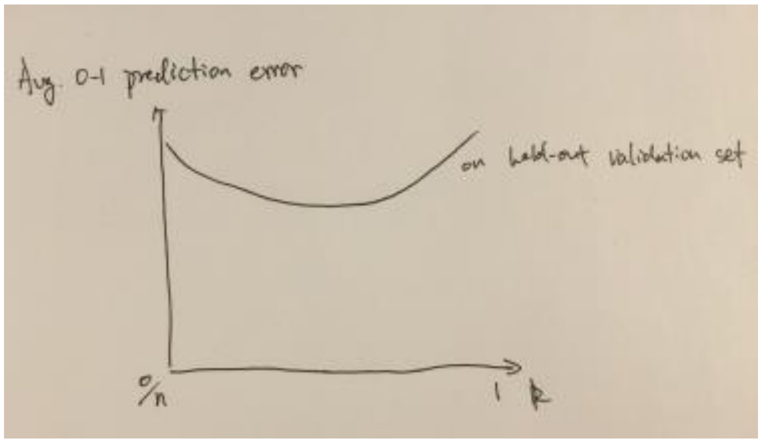

7052 new Ray Su 0.75119 4 now
Your Best Entry ↑
Your submission scored 0.75119, which is not an improvement of your best score. Keep trying!

Written Exercises

1. By definition cov[X, Y] = E[(X-E[X])(Y-E[Y])] = E[XY – X(E[Y]) – E[X]Y + E[X]E[Y]] = E[XY] – E[X]E[Y], and
   Therefore, var[X] = cov[X, X]; cov[X+Y, Z] = E[(X+Y)Z] – E[X+Y]E[Z] = E[XZ] – E[X]E[Z] + E[YZ] - E[Y]E[Z] = cov[X, Z] + cov[Y, Z]; cov[X, Y] = cov[Y, X]; cov[-X, Y] = E[-XY] – E[-X]E[Y] = -cov[X, Y]
   Thus, var[X-Y] = cov[X-Y, X-Y] = cov[X, X] + cov[-Y, -Y] + cov[X, -Y] + cov[-Y, X] = var[X] + var[Y] – 2cov[X, Y]
2. Bayes rule:

	Positive (defective) Pr(P) = 1/100,000	Negative (not defective) Pr(N) = 1 – Pr(P)
Test Positive	Pr(TP | P) = 0.95	Pr(TP | N) = 0.05
Test Negative	Pr(TN | P) = 0.05	Pr(TN | N) =0.95

   a. Pr(P | TP) = Pr(TP, P) / Pr(TP) = Pr(TP, P) / (Pr(TP, P) + Pr(TP, N)) = Pr(TP | P) * Pr(P) / (Pr(TP | P) * Pr(P) + Pr(TP | N) * Pr(N)) = 0.00019
   b. 10,000,000 * Pr(TP, N) = 10,000,000 * 0.05 * (1 – 1/100,000) = 499,995 good widgets are thrown out per year
      10,000,000 * Pr(TN, P) = 10,000,000 * 1/100,000 * 0.05 = 5 bad widgets are shipped to customers each year
3. kNN
   a. Average 0-1 prediction error on training data will converge to 0 as k varies from n to 1, because the model will be over-fitted and each training data point will always be classified as the same as itself (the nearest point)

b. 

Average 0-1 prediction error on held-out validation set will be decrease initially when k decrease from n because the model will predict every point as the same since k = n. However, after k becomes much smaller and close to 1, the model starts to over fit the training data and will lose accuracy on predicting validation set. Therefore, the overall curve would be in U-shape

c. The bigger number of folds we choose, the less bias we get towards the error rate of our system. And if there are just a small number of training data, we should get more folds, so we can use more data as the training set. However, more folds will increase the computation efficiency that may be unnecessary. Normally k is chosen between 3-10 or 50-100, but since we don't how much insight about our dataset, while there are 2 classes, to make a compromise between these two, we recommend using k = 10

d. We can put a weight on every neighbor base on the Euclidean distance – lager distance means less weight. Rather than give them a majority vote, we can get the weighted vote, which would be more accurate because points far from the target should have less impact on the prediction, just like what we had in logistic regression

e. With high dimensionality, kNN requires more computing power. More dimensions need more data entries to support the model, which are expensive to collect and process. Computation time will be longer as well.
Also, there is the curse of dimensionality. With more dimensions, L2 is no longer a good measure of distance because all data points would become more similar across the space. The model will be less useful.