

CS5785: Homework #1

Due on Wednesday, September 13, 2017

Prof. Serge Belongie

Travis Allen and Noshin Nisa

September 14, 2017

Abstract

Implementation and usage of popular machine learning algorithms on widely used datasets.

1 Introduction

In this assignment, we strengthen our understanding of machine learning algorithms and explore some of the important mathematical concepts that underly these algorithms. Our first experiment uses the MNIST dataset, a collection of images of digits; we progressively work towards implementing k-Nearest Neighbors, one of the simplest ML algorithms, in order to identify digits in an image. In our second experiment, we look at the Titanic passenger information dataset; we utilize a logistic regression algorithm in order to predict passenger survivorship, based on their information. Lastly, we complete mathematical exercises to understand concepts that were used in the derivation of the aforementioned algorithms, and develop an intuition for k-Nearest Neighbors. The procedures for each of the topics will be explained in the course of answering the related sub-questions.

2 Questions

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.*

The MNIST data downloaded in CSV format using the Pandas library [1], which allowed us to store all of the data in a DataFrame object. This data had ground-truth labels for every image that represented the digit. It also came with pixel information for all 784 pixels in the image, which indicate the shade at that pixel - [2].

(b) *Write a function to display an MNIST digit. Display one of each digit.*

Because we know that the MNIST database contains digits 0 through 9, we write a loop that goes through each of these digits; For each digit, we display the first image in the training set that has that label, and then advance to the next digit, starting again at the beginning of the training data. The matplotlib library was used to display each image. This requires images to be in the format of a two-dimensional array - [3] so we reshape our image data (excluding the label) in the appropriate format before displaying the image. These are displayed in Figure 1(b).

(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?*

The prior probability in this instance can be seen as the distribution of the probabilities of a given image (which we know is of a digit 0 through 9) will be classified as each digit, without actually examining the image data. To do this, we must base it off of the relative frequencies of each digit in the training data, which can be visualized with a histogram. We

display this histogram using `pyplots hist` function (Figure 1(c)), which allows us to specify that we want a normalized histogram (for each digit, we are given the number of instances of the digit divided by the total number of digits). The normalized histogram shows a fairly uniform distribution, as given by the similar frequencies of all the digits (around 10 percent).

(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).*

For each digit between 0 and 9 (inclusive), we choose the first image in the training set that has that digit as its label, and store both that digit and its index in separate variables; I will refer to this as our reference image and index, respectively. We then iterate through the training data and compute the L2 distance between our reference image and the training data image (using the `scipy` library's distance method - [4]). We make sure to skip the reference image itself so that it does not get compared to itself. If the distance measured is less than the previously-measured minimum distance, we store both that training image and the measured distance, and add an asterisk if the ground-truth labels for the reference image and this training image are not the same. This mismatch happened in two instances: the first 3's (the reference image) closest match was a 7, and the 6's closest match was a 0. The images of these nearest neighbors, along with their distance, are displayed in Figure 1d

(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*

To calculate these distance, every image in the training data with a label 0 or 1 has its distance computed against every other image with a label 0 or 1. If the image against which they are being compared is a genuine match (i.e. they have the same label), the distance between them is added to an array holding genuine matches. If it is not a genuine match, the distance between them is added to an array holding impostor matches. A histogram is plotted using `pyplot`: the number of genuine matches and impostor matches at each distance interval is displayed (see Figure 1(e)).

(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?*

The ROC curve we generate displays the the False Positive and True Positive rate for all of our possible distance thresholds (see figure 1(f)). The Equal Error Rate refers to the False Positive Rate at which the False Negative Rate is equal to the False Positive Rate. Because we do not store every possible distance, we choose the *first* false positive rate at which it becomes greater than the false negative rate. This occurs when the false positive rate is .24. A random classifier would have an error rate of .5.

(g) *Implement a K-NN classifier. (You cannot use external libraries for this question; it should be your own implementation.)*

The K-NN classifier functions by calculating the distance between the testing data and all

of the samples in the training data. We calculate the distances to all of the training data and sort it. After that, we find out the labels for the k-nearest neighbors (the labels corresponding to the k-first distances), and select the label that is most prevalent amongst those neighbors. We return an array of the "majority vote" winner for each sample in testing data, which act as our predictions.

(h) *Using the training data for all digits, perform 3 fold cross-validation on your K-NN classifier and report your average accuracy.*

We first use the cross_validation method from the sklearn library to generate three folds from our training data. For each fold, we calculate the accuracy that our k-NN algorithm predicts the labels for the "validation" set, after being trained on the training data. The final accuracy is the average of the accuracy for each of these folds. For our 3 fold cross-validation, the average accuracy was 96%.

(i) *Generate a confusion matrix(of size10X10)from your results. Which digits are particularly tricky to classify?*

A confusion matrix was generated for each of the 3 folds. The confusion matrix counts how often a given digit's image gets predicted as all of the possible digits. Our confusion matrix shows that 7s and 8s are often misclassified.

(j) *Train your classifier with all of the training data, and test your classifier with the test data. Submit your results to Kaggle.*

We first train our classifier and test against our test data to get an array of results (predictions) for the test data. We reformat this data for the Kaggle standards to generate a submission CSV. Our submission data predicted correctly 96.94% of the time (See figure 1(j)).

2)

(a) *Join the Titanic: Machine Learning From Disaster competition on Kaggle. Download the training and test data.*

The training and test data was downloaded, and the panda was used to store the training data csv file in a DataFrame object.

(b) *Using logistic regression, try to predict whether a passenger survived the disaster. You can choose the features (or combinations of features) you would like to use or ignore, provided you justify your reasoning.*

We first used intuition to ignore a number of columns. PassengerID, Name, and embarking location were features that seemed to have no bearing on one's survival. The ticket was complexly encoded, and we figured that the "Fare" feature encompassed most of what could be gleaned from a ticket number (because presumably, the amount that you paid influenced where you stayed and also the assistance received). Cabin was excluded because of the quality of the data: many passengers did not have cabin information. Furthermore, cabin was probably a reflection of the fare as well.

(c) *Train your classifier using all of the training data, and test it using the testing data.*

Submit your results to Kaggle.

We used sklearn's LogisticRegression class to fit our model, using our training data. We then manipulated the test data in the same way we had manipulated the training data (by dropping the columns that were dropped in part (b)) and generated an array of predictions for each test data. Submitting our results to Kaggle showed that we predicted a 75.6% accuracy (See figure 2(c)).

Written Exercises

1) *Variance of a sum*

$$E[X - Y] = E[X] - E[Y]$$

$$\text{var}(W) = E[(W - E[W])^2]$$

$$\text{Let } W = X - Y$$

$$\text{var}(X - Y) = E[((X - Y) - E[X - Y])^2]$$

$$\text{var}(X - Y) = E[((X - Y) - (E[X] - E[Y]))^2] = E[((X - E[X]) - (Y - E[Y]))^2]$$

$$\text{var}(X - Y) = E[(X - E[X])^2 + (Y - E[Y])^2 - 2(X - E[X])(Y - E[Y])]$$

$$\text{var}(X - Y) = E[(X - E[X])^2] + E[(Y - E[Y])^2] - 2E[(X - E[X])(Y - E[Y])]$$

$$\text{var}(X - Y) = \text{var}(X) + \text{var}(Y) - 2\text{cov}(X, Y)$$

2) *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 percent 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*

	Widget Functional (99.999%)	Widget Defective (.001%)
Test Positive	5%	95%
Test Negative	95%	5%

Test Positive: Comes up as defective

Test Negative: Comes up as functional

True Positive: Number of defective widgets per year that are correctly identified as defective

True Negative: Number of functional widgets per year that are correctly identified as functional

False Positive: Number of functional widgets per year that are incorrectly identified as defective

False Negative: Number of defective widgets per year that are incorrectly identified as functional

Widgets Made Per Year = 10,000,000

Defective Widgets Made Per Year = (.001%) * (10,000,000) = 100

Functional Widgets Made Per Year = (99.999%) * (10,000,000) = 9,999,900

True Positive = 100 * 95% = 95

True Negative = 9,999,900 * 95% = 9,499,905

False Positive = 9,999,900 * 5% = 499,995

$$\text{False Negative} = 100 * 5\% = 5$$

(a) Suppose the test shows that a widget is defective. What are the chances that it's actually defective given the test result?

$$\begin{aligned} &= \text{True Positive} / (\text{True Positive} + \text{False Positive}) \\ &= 95 / (95 + 499,995) \\ &= .0001899658 \\ &= 1.899658e^{-2}\% \end{aligned}$$

(b) If we throw out all widgets that are defective, how many good widgets are thrown away per year? How many bad widgets are still shipped to customers each year?

Good widgets thrown away per year is all of the false positives: **499,995**

Bad widgets shipped to customers per year is all of the false negatives: **5**

3) 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 k NN.)

The choice of k in k -Nearest Neighbors determines how many of the n samples in the training data will “vote” on the classification of the query. When k NN is performed on the training data, each training data sample is compared to the entire training set (a set that includes the sample point) and a prediction is made; the prediction error rate refers to the rate at which these points are falsely classified.

The lowest prediction error rate will occur when $k=1$: the single nearest neighbor to the query (which in this case is from the training data) will be itself, and will therefore predict the class accurately.

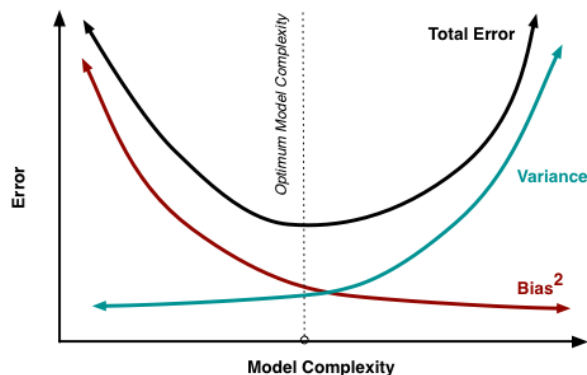
When $k=n$, the algorithm will always predict the majority that exists in the training data. The error rate will thus be equal to the prominence of the minority class in the sample, because these will always be predicted incorrectly. If $n=5$, and 3 samples have the ground-truth label of “1” and 2 have the ground-truth label of “0”, then all 5 samples will have a predicted class of “1”; this will be correct 3/5 of the time (for the 1 labels), and incorrect 2/5 of the time (for the 0 labels).

(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.

In contrast to part (a), selecting $k=1$ no longer guarantees a correct prediction: the held-out test data no longer exists in the training data, so the nearest neighbor to a test sample will not just be itself.

A low value for k will increase the error due to variance and decrease the error due to bias. Conversely, a higher value for k will increase the error due to bias, but will decrease the error due to variance.

The effects of model complexity on bias and variance, and therefore the total error, is displayed below - [5]:



In the example of k NN, increasing "model complexity" actually means reducing k , because high levels of k make the boundary between classes smoother and therefore simplifies the system - [6].

(c) *We wish to choose k by cross-validation and are considering how many folds to use. Compare both the computational requirements and the validation accuracy of using different numbers of folds for k NN and recommend an appropriate number.*

K -fold cross-validation is the process by which the parameter k (unrelated to the K of K -fold) for the k NN algorithm. The number of folds refers to the number of sub-groups the training data is divided into. One of these folds is treated as the "validation set", and k NN is performed with the remaining folds as the "training data" and the "validation set" as the test data, and an error rate is determined. This is then repeated for each of the remaining folds, and an average of the error rates is computed. By comparing the average error rates for a given choice of parameter k allows us to assess which k to choose.

The choice of the number of folds comes with its own set of trade-offs. Because a model is fit for each of the folds, increasing the number of folds makes it more computationally expensive. However, increasing the number of folds also increases the accuracy of the error prediction, because the error rate is performed more times. The most extreme example would be Leave-One-Out Cross-Validation, in which the number of folds is simply the number of samples in the training set; this has the highest accuracy but is extremely expensive, computationally.

In many instances, 10 folds offers the best trade-off between computational speed and the accuracy. In the absence of more information on the training set itself, this widely-used number of folds appears to be a good choice - [8].

(d) *In k NN, 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.*

Equally weighted votes from neighbors can have negative consequences when k is large, be-

cause in that algorithm, a very distant neighbor has the same influence on the final decision as the closest neighbor. A potential alteration to that algorithm would be to weight the neighbors in proportion to their proximity: The first neighbor would have a weight of 1, the second neighbor $1/2$, etc. until the k -neighbor, which would have a weight of $1/k$.

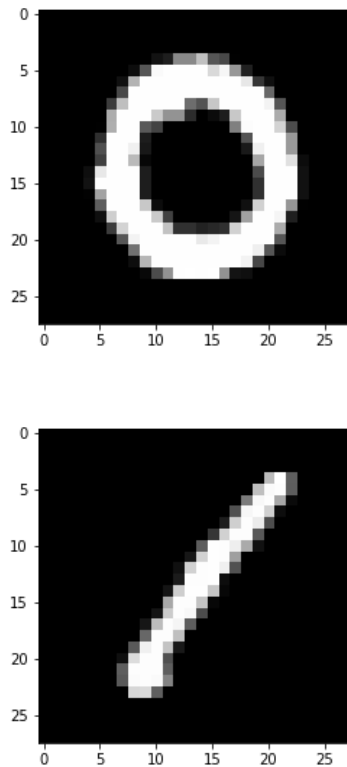
(e) *Give two reasons why kNN may be undesirable when the input dimension is high.*

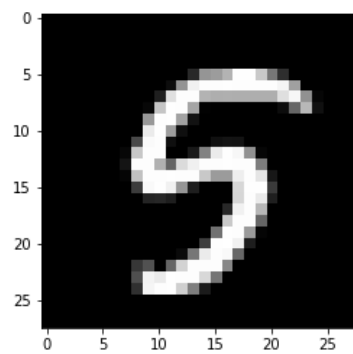
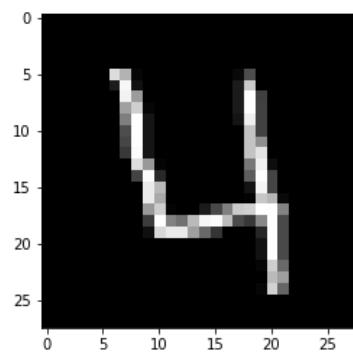
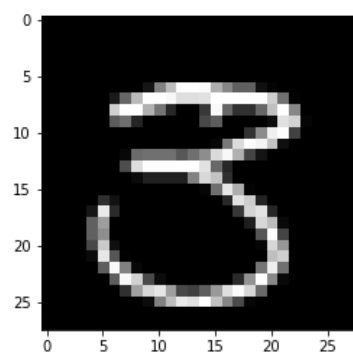
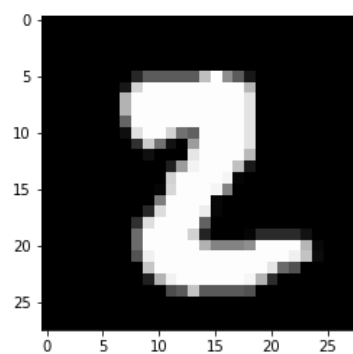
kNN requires a computation of distance between the query and each of the samples in the training data. When input dimension is high, the computation of this distance becomes more complex and makes the algorithm more inefficient.

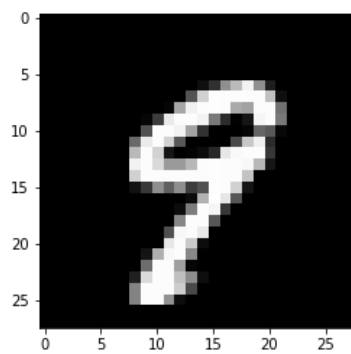
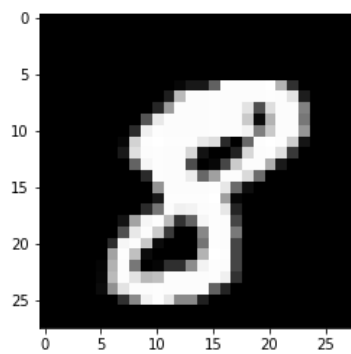
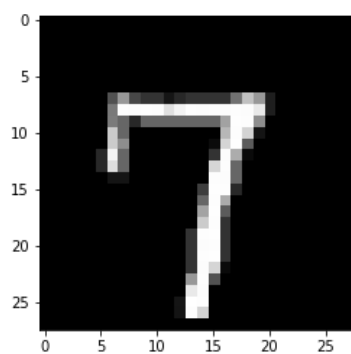
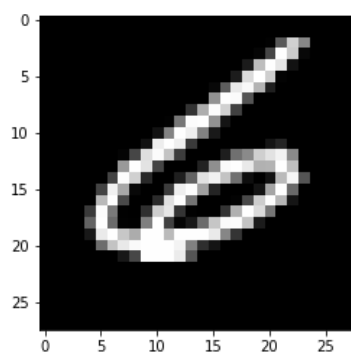
Furthermore, for a given n , as you increase the dimensionality, the data becomes more sparse (a data point and its neighbor are further apart in space). As a result, the distance from a data point's closest neighbor and its furthest neighbor begin to converge, making the concept of a "nearest neighbor" less relevant - [7]

3 Figures

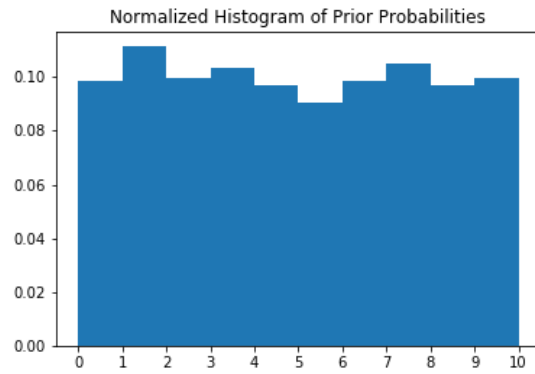
Digit Recognizer: 1(b)





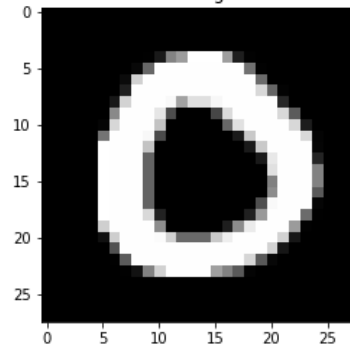


Digit Recognizer: 1(c)

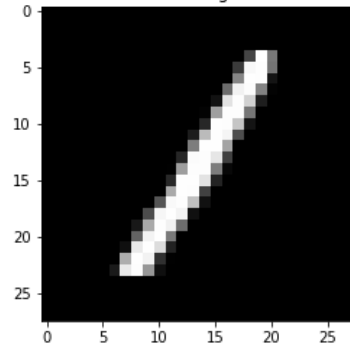


Digit Recognizer: 1(d)

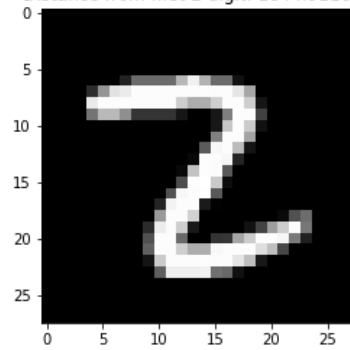
Distance from first 0 digit: 1358.31881383



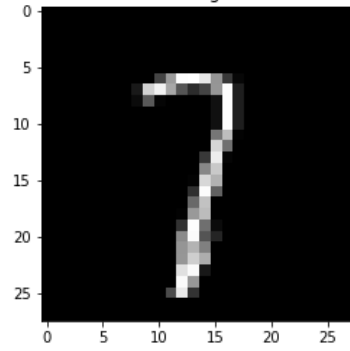
Distance from first 1 digit: 968.81886852



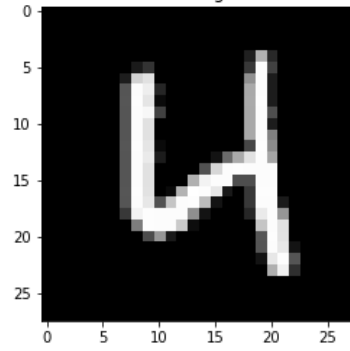
Distance from first 2 digit: 1844.911651



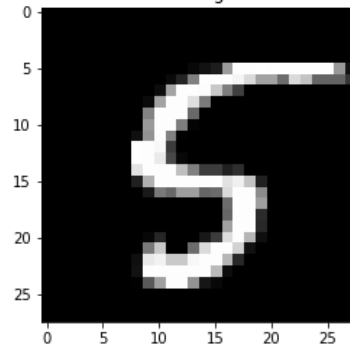
Distance from first 3 digit: 2120.22711048*



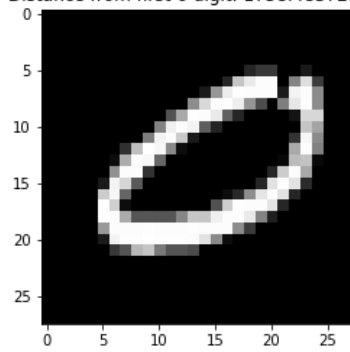
Distance from first 4 digit: 1745.39708949

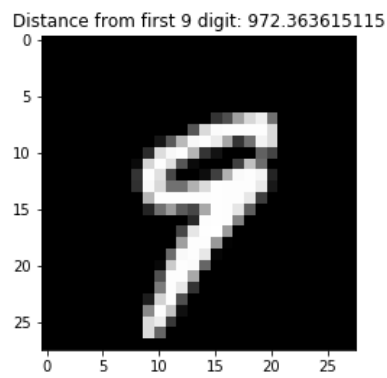
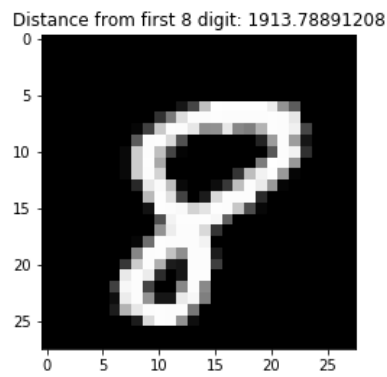
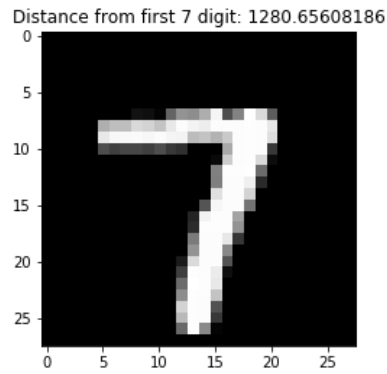


Distance from first 5 digit: 1600.40432391

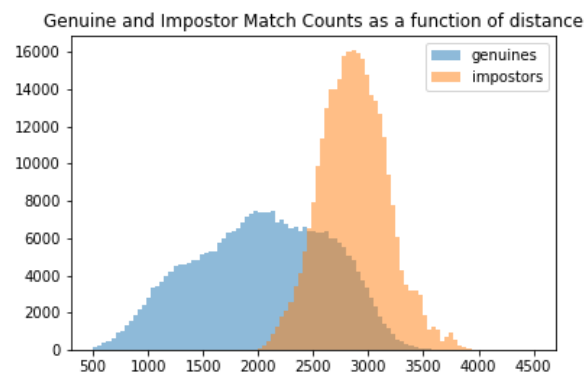


Distance from first 6 digit: 1758.48372185*

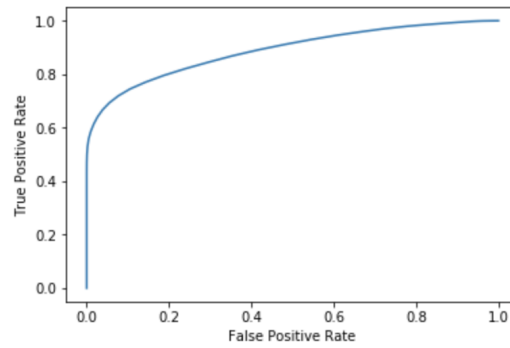




Digit Recognizer: 1(e)





Digit Recognizer: 1(f)





Digit Recognizer: 1(i)

```
[[ 1358 0 2 0 0 2 7 0 1 1]
[ 0 1561 1 2 2 1 4 1 1 2]
[ 7 17 1364 6 2 1 1 22 4 3]
[ 0 2 6 1369 0 12 1 7 9 4]
[ 0 16 0 0 1308 0 5 3 0 36]
[ 3 0 1 20 0 1227 11 2 3 9]
[ 6 1 0 0 3 8 1381 0 0 0]
[ 0 22 2 0 7 0 0 1414 0 19]
[ 5 11 5 20 1 30 8 4 1240 19]
[ 2 3 0 7 13 4 2 23 4 1309]]
[[ 1339 0 2 0 0 3 4 0 1 1]
[ 0 1517 5 0 1 1 1 6 1 0]
[ 8 18 1329 5 1 0 3 29 4 0]
[ 1 1 9 1438 0 18 1 11 10 10]
[ 2 14 0 0 1325 0 4 3 1 39]
[ 3 0 0 22 1 1189 16 1 2 11]
[ 8 2 0 0 1 9 1358 0 2 0]
[ 0 19 3 2 1 0 0 1417 0 18]
[ 2 10 5 21 8 26 6 2 1256 15]
[ 4 3 1 12 16 3 0 12 5 1342]]
[[ 1408 0 0 0 0 1 2 0 0 0]
[ 0 1573 2 0 1 0 0 1 0 0]
[ 14 6 1295 4 0 2 3 25 2 2]
[ 1 6 14 1388 0 14 0 7 7 5]
[ 0 12 0 0 1262 0 7 1 0 34]
[ 5 2 0 20 2 1213 19 1 4 8]
[ 5 2 0 0 2 5 1344 0 0 0]
[ 1 12 3 0 3 0 0 1441 0 17]
[ 5 25 4 16 5 22 6 4 1266 16]
[ 6 3 2 10 10 4 0 26 3 1359]]
```

Digit Recognizer: 1(j)

1006	new	NoshinNisa		0.96942	1	now
Your Best Entry 						
Your submission scored 0.96942, which is not an improvement of your best score. Keep trying!						

Titanic: 2(c)

6727	new	NoshinNisa		0.75598	1	19h
Your Best Entry 						
Your submission scored 0.75598, which is not an improvement of your best score. Keep trying!						

4 Source Code

Digit Recognizer

```
1
2 # coding: utf-8
3
4 #
```

```

5
6 # In [120]:
7
8 get_ipython().magic(u'matplotlib inline')
9 import numpy as np
10 import pandas as pd
11 import scipy.misc as smp
12 import scipy.spatial.distance as dist
13 from sklearn.metrics.pairwise import euclidean_distances
14 import matplotlib.pyplot as plt
15 from collections import Counter
16 from sklearn import cross_validation
17 from sklearn.metrics import confusion_matrix
18
19 trainingData = pd.read_csv('train.csv')
20 testingData = pd.read_csv('test.csv')
21 noLabels = trainingData.loc[:, 'pixel0': 'pixel783']
22 onlyLabels = trainingData.loc[:, 'label'].values
23 labels = trainingData['label']
24 testingData
25
26
27 # In [7]:
28
29 ## 1b
30 def displayDigit(noLabelImageData, classification):
31     imageData = noLabelImageData.values.reshape([28,28])
32     plt.figure()
33     plt.gray()
34     plt.imshow(imageData)
35
36 for number in range(0,10):
37     for index, data in enumerate(trainingData):
38         if (trainingData.iloc[index].label == number):
39             displayDigit(noLabels.iloc[index], number)
40             break
41
42
43
44
45
46
47 # In [8]:
48
49 # (1c)
50
51 xs = [0,1,2,3,4,5,6,7,8,9,10]
52 http://matplotlib.org/api/pyplot\_api.html#matplotlib.pyplot.xticks
53 plt.xticks(xs)
54 https://matplotlib.org/devdocs/api/\_as\_gen/matplotlib.pyplot.hist.html
55 plt.hist(onlyLabels, normed=True, bins=xs)
56
57
58

```

```

59 # In [9]:
60
61 #(1d)
62
63 for number in range(0,10):
64     minDistance = float("inf")
65     # get the first row with the label of number
66     firstRowOfNumber = trainingData[trainingData['label'] == number].iloc[0]
67     indexOfSelectedRow = trainingData.index.get_loc(firstRowOfNumber.name) #
68     # get Index so we can skip it when we compare to training data
69     for index, data in enumerate(trainingData):
70         if (index == indexOfSelectedRow):
71             continue #skip the image being compared, bc this would yield
72             distance of 0
73         else:
74             noLabelFirstRow = noLabels.iloc[indexOfSelectedRow]
75             noLabelTrainingDataSample = noLabels.iloc[index]
76             dst = dist.euclidean(noLabelFirstRow, noLabelTrainingDataSample)
77             # https://docs.scipy.org/doc/scipy/reference/generated/scipy.
78             spatial.distance.euclidean.html#scipy.spatial.distance.euclidean
79             if dst < minDistance:
80                 minDistance = dst
81                 bestMatch = noLabelTrainingDataSample
82                 if trainingData.iloc[index].label == trainingData.iloc[
83                     indexOfSelectedRow].label:
84                     appendString = "" # will append nothing if labels match
85                 else:
86                     appendString = "*" # will append asterisk if labels
87                     mismatch
88             imageDataForNearestNeighbor = bestMatch.values.reshape([28,28])
89             plt.figure()
90             plt.gray()
91             plt.title("Distance from first " + str(number) + " digit: " + str(
92                 minDistance) + appendString)
93             plt.imshow(imageDataForNearestNeighbor)
94
95 #
96
97 # In [10]:
98
99 #1(e)
100 genuineMatches = []
101 impostorMatches = []
102
103 onlyOnesAndZeroes = trainingData[(trainingData['label'] == 0) | (trainingData[
104     'label'] == 1)]
105
106 for FirstIndex, data in enumerate(onlyOnesAndZeroes):
107     for SecondIndex, secondData in enumerate(onlyOnesAndZeroes):

```



```

105     distance = dist.euclidean(onlyOnesAndZeroes.iloc[FirstIndex],
106                                onlyOnesAndZeroes.iloc[SecondIndex])
107
108     if onlyOnesAndZeroes.iloc[FirstIndex].label == 0 and onlyOnesAndZeroes
109        .iloc[SecondIndex].label == 0:
110         genuineMatches.append(distance)
111     if onlyOnesAndZeroes.iloc[FirstIndex].label == 1 and onlyOnesAndZeroes
112        .iloc[SecondIndex].label == 1:
113         genuineMatches.append(distance)
114     if onlyOnesAndZeroes.iloc[FirstIndex].label == 1 and onlyOnesAndZeroes
115        .iloc[SecondIndex].label == 0:
116         impostorMatches.append(distance)
117     if onlyOnesAndZeroes.iloc[FirstIndex].label == 0 and onlyOnesAndZeroes
118        .iloc[SecondIndex].label == 1:
119         impostorMatches.append(distance)
120
121 # In [11]:
122
123 bins = np.linspace(500, 4500, 100)
124 plt.hist(genuineMatches, bins, alpha=0.5, label='genuines')
125 plt.hist(impostorMatches, bins, alpha=0.5, label='impostors')
126 plt.legend(loc='upper right')
127 plt.show()
128
129 # In [12]:
130
131 #1(f)
132 def truePosFalsePosRates(threshold):
133     truePos=0
134     flaseNeg=0
135     falsePos=0
136     trueNeg=0
137
138     for i in genuineMatches:
139         if i < threshold:
140             truePos=truePos+1
141         else:
142             flaseNeg=flaseNeg+1
143
144     for i in impostorMatches:
145         if i < threshold:
146             falsePos=falsePos+1
147         else:
148             trueNeg=trueNeg+1
149
150     falsePosRate=(falsePos)/(falsePos+trueNeg)
151     truePosRate=(truePos)/(truePos+flaseNeg)
152     return truePosRate, falsePosRate
153
154 x=[]
155 y=[]
156 threshold = []

```

```

154 for i in range(0,3995,50):
155     threshold.append(i)
156     truePosRate,falsePosRate = truePosFalsePosRates(i)
157     x.append(falsePosRate)
158     y.append(truePosRate)
159
160 plt.plot(x,y)
161 plt.ylabel('True Positive Rate')
162 plt.xlabel('False Positive Rate')
163 plt.show()
164
165
166 # In [13]:
167
168 l = [0,1]
169 m = [1,0]
170 plt.plot(l,m)
171 plt.plot(x,y)
172 plt.ylabel('True Positive Rate')
173 plt.xlabel('False Positive Rate')
174 plt.show()
175
176
177 # In [34]:
178
179 err = 0.0
180 for index, data in enumerate(y):
181     if (1 - y[index]) < x[index] and y[index] > 0 and x[index] > 0:
182         err_value = x[index]
183         break
184
185 print (err_value)
186
187
188 # In [75]:
189
190 #l(g)
191
192 #https://docs.python.org/3/library/collections.html#collections.Counter
193
194 def match(k_neighbors):
195     counter = Counter(k_neighbors)
196     max_count = max(counter.values())
197     match = [k for k,v in counter.items() if v == max_count]
198     return match[0]
199
200 def all_k_neighbors(sortedValues, k, labels):
201     result = []
202     for i in sortedValues:
203         label_list = [labels[x] for x in list(i[:k])]
204         result.append(match(label_list))
205     return result
206
207 def KNN (testingData, trainingData, k, labels):

```

```

208     distance = euclidean_distances(testingData, trainingData)
209     sortedValues = np.argsort(distance)
210     return all_k_neighbors(sortedValues, k, labels)
211
212
213 #https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
214
215
216 # In [81]:
217
218 #l(h)
219 three_folds = cross_validation.KFold(len(noLabels), n_folds=3)
220
221
222 aveAccuracy = 0
223 for i in three_folds:
224     train, test = i
225
226     X = noLabels.as_matrix()
227     Y = trainingData.as_matrix()
228     Y_train = labels.as_matrix()
229
230     x_train = [X[x] for x in list(train)] #nolabels
231     y_train = [Y_train[x] for x in list(train)] #labels
232
233     x_test = [X[x] for x in list(test)]
234     y_test = [Y_train[x] for x in list(test)]
235
236     validation = KNN(x_test, x_train, 3, y_train)
237     aveAccuracy = aveAccuracy + np.mean((np.asarray(validation)==y_test))
238     print (confusion_matrix(y_test, validation))
239
240 aveAccuracy = aveAccuracy/3
241 print aveAccuracy
242
243
244 # In [82]:
245
246 testingData = testingData.iloc[:, :]
247 result = KNN(testingData, noLabels, 3, labels)
248 result
249
250
251 # In [121]:
252
253 digitResult = pd.DataFrame({"Label": result})
254
255 #https://stackoverflow.com/questions/18176933/create-an-empty-data-frame-with-
    index-from-another-data-frame
256 #https://stackoverflow.com/questions/18022845/pandas-index-column-title-or-
    name
257
258 digitResult.insert(0, "ImageId", testingData.index+1)

```

```
259 digitResult.to_csv("submission.csv",index=False)
```

Listing 1: Digit Recognizer Source Code

Titanic Disaster

```
1
2 # coding: utf-8
3
4 # In [29]:
5
6 import numpy as np
7 import matplotlib.pyplot as plt
8 from sklearn import datasets, linear_model
9 from sklearn.linear_model import LogisticRegression
10 from sklearn.model_selection import train_test_split
11 import pandas as pd
12
13 dataset = pd.read_csv('train.csv')
14 dataset.head(10)
15
16
17 # In [30]:
18
19 dataset=dataset.drop(["PassengerId", "Name", "Ticket", "Cabin", "Embarked"],
20                      axis=1)
21 dataset.head(10)
22
23 # In [31]:
24
25 dataset.fillna(value=0, inplace=True)
26
27
28 # In [32]:
29
30 dataset
31
32
33 # In [33]:
34
35 dataset['Sex'] = dataset['Sex'].map({'female': 1, 'male': 0})
36 dataset
37
38
39 # In [37]:
40
41 X = dataset[["Pclass","Sex","Age", "Parch", "SibSp", "Fare"]]
42 Y = dataset[["Survived"]]
43 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4)
44
45
46 # In [38]:
47
48 Reg = LogisticRegression()
```

```

49 Reg.fit(X_train, Y_train)
50
51
52 # In [40]:
53
54 Reg.fit(X_train, Y_train).score(X_test, Y_test)
55
56
57 # In [77]:
58
59 #Loading testing dataset and preparing it for prediction
60 TestDataset = pd.read_csv('test.csv')
61 resultIndex = TestDataset[["PassengerId"]]
62 TestDataset = TestDataset.drop(["PassengerId", "Name", "Ticket", "Cabin", "
    Embarked"], axis=1)
63 TestDataset.fillna(value=0, inplace=True)
64 TestDataset['Sex'] = TestDataset['Sex'].map({'female': 1, 'male': 0})
65 TestX = TestDataset[["Pclass", "Sex", "Age", "Parch", "SibSp", "Fare"]]
66 TestDataset
67
68
69 # In [78]:
70
71 predictions = Reg.predict(TestX)
72
73
74 # In [79]:
75
76 predictions
77
78
79 # In [92]:
80
81 survivedResult = pd.DataFrame({"Survived": predictions})
82 predictionResult = pd.concat([resultIndex, survivedResult], axis=1)
83 predictionResult
84 predictionResult.to_csv("submission.csv", index=False)

```

Listing 2: Titanic Source Code

5 Analysis

Our experiments with the kNN algorithm exposed us both to the advantages and drawbacks of the algorithm. Our 96% accuracy in prediction of MNIST digits demonstrated how effective some implementations of it can be on relatively simple classification problems. However, the conceptual exercises showed us that for high-dimensional data it can be slow and inaccurate, and that the lack of weighted voting (which is the case for the simplest implementation) can be ineffective for larger values of k . Our experiment with the Titanic disaster database introduced the process of feature engineering, in which we identify features needed to make algorithms work (which in this case, was a logistic regression algorithm). The fact that our accuracy was higher when tested against a sub-set of our training data that was held-out

during training as compared to our accuracy for the external test data, suggests that we may have over-fit our model to the training data.

References

- [1] *Pandas library*, available at <http://pandas.pydata.org/>
- [2] *Kaggle Dataset explanation*, available at <https://www.kaggle.com/c/digit-recognizer/data>
- [3] *Matplotlib's Pyplot documentation*, https://matplotlib.org/api/pyplot_api.html
- [4] *SciPy library* found at <https://docs.scipy.org/doc/scipy/reference/spatial.distance.html>
- [5] *Sketch of error and model complexity*, found at <http://scott.fortmann-roe.com/docs/BiasVariance.html>
- [6] *Relationship between k and model complexity*, found at <https://stackoverflow.com/questions/23767820/why-does-decreasing-k-in-k-nearest-neig>
- [7] *Conversation of curse of dimensionality* found at <https://stats.stackexchange.com/questions/65379/machine-learning-curse-of-dimensiona>
- [8] *Fold number choice*, found at <https://stats.stackexchange.com/questions/27730/choice-of-k->