ECE 4424 Homework Set #1 Jamahl Savage

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

Problem 1: Supervised Learning

I believe this type of problem could be a binary classification problem due to the fact that we are trying to analyze whether the company's current customers have cancelled their subscription or not. The data we might need for this setup would include: Customer ID, user preferences, how much time a user spends using the service, their subscription plan, the date, time and area the user is using the service. The label space would be whether the customer has churned out or not (true or false). Possible loss functions that could be used for this classification problem would be the mean square loss or hinge loss functions. The hypothesis space would depend on the chosen algorithm and the configuration of the algorithm which can then be used to approximate the target function.

The company should use the results to provide a better user experience to maintain its current users/ minimize its customer churn rate. This could be done by better suggesting more accurate recommendations based on a user's watch history, suggesting a user through push notifications to check out a show during a time when they would most likely be using the service, or by offering appealing subscription renewal deals.

Problem 2: K-Nearest Neighbor

Bag-of-words								
ID	money	free	for	gambling	fun	machine	learning	spam
1	3	0	0	0	0	0	0	True
2	1	2	1	1	1	0	0	True
3	0	0	1	1	1	0	0	True
4	0	0	1	0	3	1	1	False
5	0	1	0	0	0	1	1	False
Query	0	1	1	0	0	1	1	?

Euclidean distance $= \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$

$x = (a,b)$ $y = (c,d)$

Manhattan Distance $= \sum |a-c| + |b-d|$

ID 1 & Query:

money: $(0-3)^2 = 9$

free: $(1-0)^2 = 1$

for: $(1-0)^2 = 1$

gambling: $(0-0)^2 = 0$

fun: $(0-0)^2 = 0$

machine: $(1-0)^2 = 1$

learning: $(1-0)^2 = 1$

$|0-3| = 3$

$|1-0| = 1$

$|1-0| = 1$

$|0-0| = 0$

$|0-0| = 0$

$|1-0| = 1$

$|1-0| = 1$

Euclidean distance $= \sqrt{9+1+1+1+1} = 3.61$

Manhattan Distance $= 7$

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

ID 2 & Query:

money: $(0-1)^2 = 1$ $|0-1| = 1$
free: $(1-2)^2 = 1$ $|1-2| = 1$
for: $(1-1)^2 = 0$ $|1-1| = 0$
gambling: $(0-1)^2 = 1$ $|0-1| = 1$
fnn: $(0-1)^2 = 1$ $|0-1| = 1$
machine: $(1-0)^2 = 1$ $|1-0| = 1$
learning: $(1-0)^2 = 1$ $|1-0| = 1$

Euclidean distance = $\sqrt{1+1+1+1+1+1} = 2.45$

Manhattan Distance = 6

ID 3 & Query:

money: $(0-0)^2 = 0$ $|0-0| = 0$
free: $(1-0)^2 = 1$ $|1-0| = 1$
for: $(1-1)^2 = 0$ $|1-1| = 0$
gambling: $(0-1)^2 = 1$ $|0-1| = 1$
fnn: $(0-1)^2 = 1$ $|0-1| = 1$
machine: $(1-0)^2 = 1$ $|1-0| = 1$
learning: $(1-0)^2 = 1$ $|1-0| = 1$

Euclidean distance = $\sqrt{1+1+1+1+1} = 2.24$

Manhattan Distance = 5

ID 4 & Query:

money: $(0-0)^2 = 0$ $|0-0| = 0$
free: $(1-0)^2 = 1$ $|1-0| = 1$
for: $(1-1)^2 = 0$ $|1-1| = 0$
gambling: $(0-0)^2 = 0$ $|0-0| = 0$
fnn: $(0-3)^2 = 9$ $|0-3| = 3$
machine: $(1-1)^2 = 0$ $|1-1| = 0$
learning: $(1-1)^2 = 0$ $|1-1| = 0$

Euclidean distance = $\sqrt{1+9} = 3.16$

Manhattan Distance = 4

ECE 4424 Homework Set #1 Jamahl Savage
Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

ID 5 & Query:

money: $(0-0)^2 = 0$ $|0-0| = 0$
free: $(1-1)^2 = 0$ $|1-1| = 0$
for: $(1-0)^2 = 1$ $|1-0| = 1$
gambling: $(0-0)^2 = 0$ $|0-0| = 0$
fun: $(0-0)^2 = 0$ $|0-0| = 0$
machine: $(1-1)^2 = 0$ $|1-1| = 0$
learning: $(1-1)^2 = 0$ $|1-1| = 0$

Euclidean distance = $\sqrt{1} = 1$

Manhattan Distance = 1

Bag-of-words									
ID	money	free	for	gambling	fun	machine	learning	spam	ED; MD
1	3	0	0	0	0	0	0	True	3.61; 7
2	1	2	1	1	1	0	0	True	2.45; 6
3	0	0	1	1	1	0	0	True	2.24; 5
4	0	0	1	0	3	1	1	False	3.16; 4
5	0	1	0	0	0	1	1	False	1; 1
Query	0	1	1	0	0	1	1	?	

Bag-of-words									
ID	money	free	for	gambling	fun	machine	learning	spam	Vector Length
1	3	0	0	0	0	0	0	True	3
2	1	2	1	1	1	0	0	True	2.83
3	0	0	1	1	1	0	0	True	1.73
4	0	0	1	0	3	1	1	False	3.46
5	0	1	0	0	0	1	1	False	1.73
Query	0	1	1	0	0	1	1	?	2

Dot Product to Query:

ID 1 & Query:

$(3\times0)+(0\times1)+(0\times1)+(0\times0)+(0\times0)+(0\times1)+(0\times1)$

$= 0$

ECE 4424 Homework Set #1 Jamahl Savage

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

ID 2 & Query:

$$(1\times0)+(2\times1)+(1\times1)+(1\times0)+(1\times0)+(0\times1)+(0\times1)$$

$$=3$$

ID 3 & Query:

$$(0\times0)+(0\times1)+(1\times1)+(1\times0)+(1\times0)+(0\times1)+(0\times1)$$

$$=1$$

ID 4 & Query:

$$(0\times0)+(0\times1)+(1\times1)+(0\times0)+(3\times0)+(1\times1)+(1\times1)$$

$$=3$$

ID 5 & Query:

$$(0\times0)+(1\times1)+(0\times1)+(0\times0)+(0\times0)+(1\times1)+(1\times1)$$

$$=3$$

Cosine Similarity: $\text{Cosine Similarity} = \frac{X \cdot Y}{\|X\|\|Y\|}$

ID 1 & Query:

$$\frac{0}{2\times3}=0$$

ID 2 & Query:

$$\frac{3}{2\times2.83}=0.53$$

ID 3 & Query:

$$\frac{1}{2\times1.73}=0.29$$

ID 4 & Query:

$$\frac{3}{2\times3.46}=0.43$$

ID 5 & Query:

$$\frac{3}{2\times1.73}=0.87$$

ECE 4424 Homework Set #1 Jamahl Savage

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

Bag-of-words									
ID	money	free	for	gambling	fun	machine	learning	spam	Cosine Similarity
1	3	0	0	0	0	0	0	True	0
2	1	2	1	1	1	0	0	True	0.53
3	0	0	1	1	1	0	0	True	0.29
4	0	0	1	0	3	1	1	False	0.43
5	0	1	0	0	0	1	1	False	0.87
Query	0	1	1	0	0	1	1	?	

Using the Mean Absolute Error Distance function: $d(x,y) = \frac{1}{n}\sum_{i=1}^{n} |x_i - y_i|$

ID 1 & Query:

$\frac{1}{7}(7) = 1$

$|0-3| = 3$

$|1-0| = 1$

$|1-0| = 1$

$|0-0| = 0$

$|0-0| = 0$

$|1-0| = 1$

$|1-0| = 1$

ID 2 & Query:

$\frac{1}{7}(6) = 0.86$

$|0-1| = 1$

$|1-2| = 1$

$|1-1| = 0$

$|0-1| = 1$

$|0-1| = 1$

$|1-0| = 1$

$|1-0| = 1$

ECE 4424 Homework Set #1 Jamahl Savage

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

ID 3 & Query:

$\frac{1}{7}(5) = 0.71$

$|0-0| = 0$
$|1-0| = 1$
$|1-1| = 0$
$|0-1| = 1$
$|0-1| = 1$
$|1-0| = 1$
$|1-0| = 1$

ID 4 & Query:

$\frac{1}{7}(4) = 0.57$

$|0-0| = 0$
$|1-0| = 1$
$|1-1| = 0$
$|0-0| = 0$
$|0-3| = 3$
$|1-1| = 0$
$|1-1| = 0$

ID 5 & Query:

$\frac{1}{7}(1) = 0.14$

$|0-0| = 0$
$|1-1| = 0$
$|1-0| = 1$
$|0-0| = 0$
$|0-0| = 0$
$|1-1| = 0$
$|1-1| = 0$

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

Bag-of-words									
ID	money	free	for	gambling	fun	machine	learning	spam	MAE
1	3	0	0	0	0	0	0	True	1
2	1	2	1	1	1	0	0	True	0.86
3	0	0	1	1	1	0	0	True	0.71
4	0	0	1	0	3	1	1	False	0.57
5	0	1	0	0	0	1	1	False	0.14
Query	0	1	1	0	0	1	1	?	

a. False; NOT spam
b. True; Spam
c. False; NOT spam
d. False; NOT Spam
e. Nearest neighbors are IDs 5, 4, and 3 so result is False; NOT spam

Problem 3: Probability and Linear Algebra Review

3.1.

$P(E)$

$E = \{(1,1), (1,3), (1,5), (2,2), (2,4), (2,6), (3,1), (3,3), (3,5), (4,2), (4,4), (4,6), (5,1), (5,3), (5,5), (6,2), (6,4), (6,6)\}$

$\frac{18}{36 \text{ possible rolls}} = P(E) = 0.5$

$P(F)$

$F = \{(1,6), (2,6), (3,6), (4,6), (5,6), (6,6), (6,1), (6,2), (6,3), (6,4), (6,5)\}$

$P(F) = \frac{11}{36} = 0.31$

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

P(G)

$G = \{(1,1), (2,2), (3,3), (4,4), (5,5), (6,6)\}$

$\frac{6}{36} = \frac{1}{6} = 0.17$

P(E U F)

18 + 11 - (5 similarities) = 24

$P(E \cup F) = \frac{24}{36} = \frac{2}{3} = 0.67$

P(E ∩ F)

Only 5 similarities

$P(E \cap F) = \frac{5}{36} = 0.14$

P(F U G)

11 + 6 - (1 similarity) = 16

$P(F \cup G) = \frac{16}{36} = \frac{4}{9} = 0.44$

P(F ∩ G)

$P(F \cap G) = \frac{1}{36} = 0.03$

ECE 4424 Homework Set #1 Jamahl Savage
Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

3.2.

36 Possible Outcomes

$P(A) = \{(1,1),(2,2),(3,3),(4,4),(5,5),(6,6)\} = \frac{6}{36} = \frac{1}{6}$

$P(D) = \{(1,2),(2,1),(2,3),(3,2),(3,4),(4,3),(4,5),(5,4),(5,6),(6,5)\}$
$= \frac{10}{36} = \frac{5}{18}$

16 out of 36 possible outcomes

20 out of 36 possible outcomes in which neither a doubles nor pairs where the dice differ by 1. $\frac{20}{36} = \frac{5}{9}$

$\frac{1}{6}$ chance of a pair occuring

$P(E_n) = \left(\frac{5}{9}\right)^{n-1} \frac{1}{6}$

$P(E) = \sum_{n=1}^{\infty} P(E_n) = \sum_{n=1}^{\infty} \left(\frac{5}{9}\right)^{n-1} \frac{1}{6}$ (Geometric Series) $\sum_{n=1}^{\infty} a(r)^{n-1} = \frac{a}{1-r}$

$= \frac{\frac{1}{6}}{1-\left(\frac{5}{9}\right)} = 0.375$ $P(E) = 0.375$

3.3.

Probability of picking from Urn A or B = 1/2
Probability of picking red from Urn A = 99/100

$P\left(\begin{matrix}\text{red} \\ \text{from} \\ A\end{matrix}\right) = \frac{(1/2)\left(\frac{99}{100}\right)}{\left[(1/2)(99/100) + (1/2)(1/100)\right]} = \frac{99}{100} = 0.99$

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

3.4. $A = U \Sigma V^T$

Orthogonal $\Rightarrow A^T A = A A^T = n \times n$ (symmetric matrix)

Therefore $A^T A$ will be diagonal with entries $\sigma_1^2 \dots \sigma_n^2$

The eigenvalues of the diagonal matrix $A^T A$ are the columns of I, so $V = I$ (identity matrix) in the SVD.

The unit eigenvectors are $\frac{A u_i}{\sigma_i} = \frac{w_i}{\sigma_i}$

Thus, $A = U \Sigma V^T = (A \Sigma^{-1})(\Sigma)(I)$

Problem 4: Programming Assignment – KNN

Task 1:

```
## Computes squared Euclidean distance between two vectors.
import numpy as np
def eucl_dist(x,y):
    # input:
    # x, y: vectorization of an image
    # output:
    # the euclidean distance between the two vectors

    ### STUDENT: YOUR CODE HERE
    #distance = np.linalg.norm(np.array(x) - np.array(y))
    distance = np.sqrt(np.sum((x-y)**2))
    return distance
    ### CODE ENDS
```

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

Task 2:

```
# Take a vector x and returns the indices of its K nearest neighbors in the training set: train_data

import operator

def find_KNN(x, train_data, train_labels, K, dist=eucl_dist):

    # Input:

    # x: test point

    # train_data: training data X

    # train_labels: training data labels y

    # K: number of nearest neighbors considered

    # dist: default to be the eucl_dist that you have defined above

    # Output:

    # The indices of the K nearest neighbors to test point x in the training set


    ##### STUDENT: Your code here #####

    distances = []


    for index in range(len(train_data)):

        distance = dist(x, train_data[index])

        distances.append([index, distance, train_labels[index]])

    distances.sort(key=operator.itemgetter(1))


    neighbors = []


    for i in range(K):

        neighbors.append(distances[i][0])


    return neighbors


    ##### END OF CODE #####
```

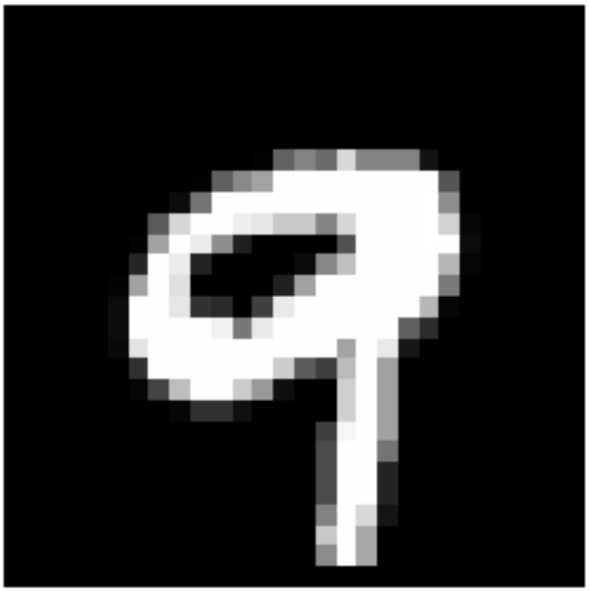
Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

```
# KNN classification

def KNN_classifier(x, train_data, train_labels,K,dist=eucl_dist):

    # Input:

    # x: test point

    # train_data: training data X

    # train_labels: training data labels y

    # K: number of nearest neighbors considered

    # dist: default to be the eucl_dist that you have defined above

    # Output:

    # the predicted label of the test point


    ##### STUDENT: Your code here #####

    neighbors = find_KNN(x, train_data, train_labels, K, dist=eucl_dist)

    #print(neighbors)

    output_vals = []

    for i in range(len(neighbors)):

        output_vals.append(train_labels[neighbors[i]])


    prediction = max(set(output_vals), key=output_vals.count)


    return int(prediction)

    ##### END OF CODE #####
```

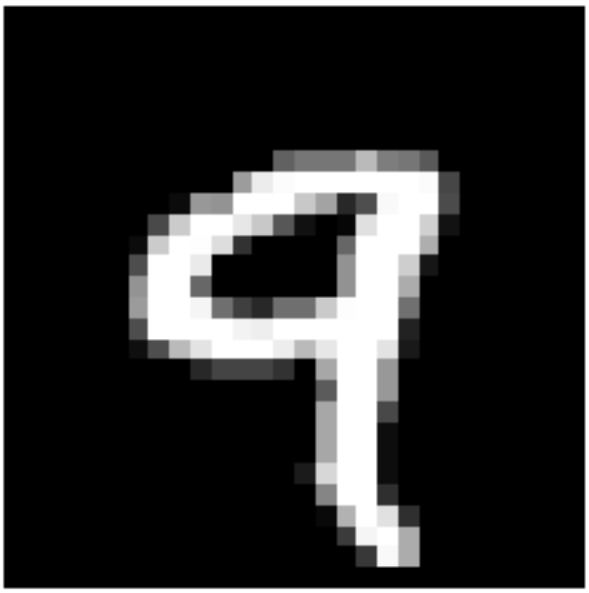
Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

Task 3:

```
A success case:
1-NN classification:  9
True label:  9
The test image:
```

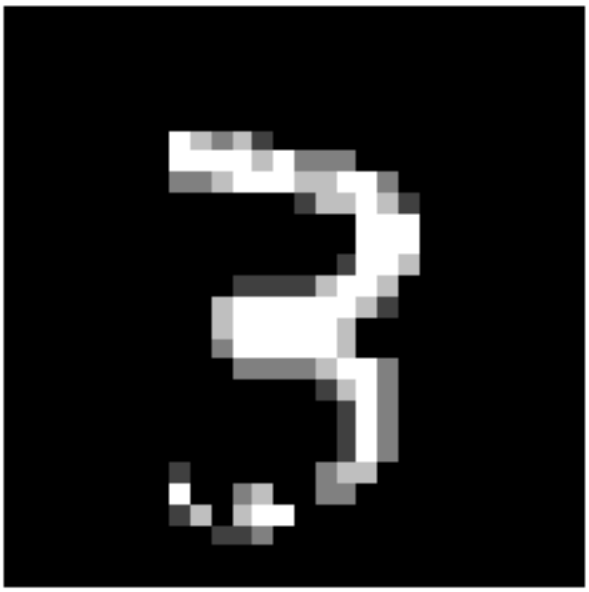


```
Label 9
The corresponding nearest neighbor image:
```



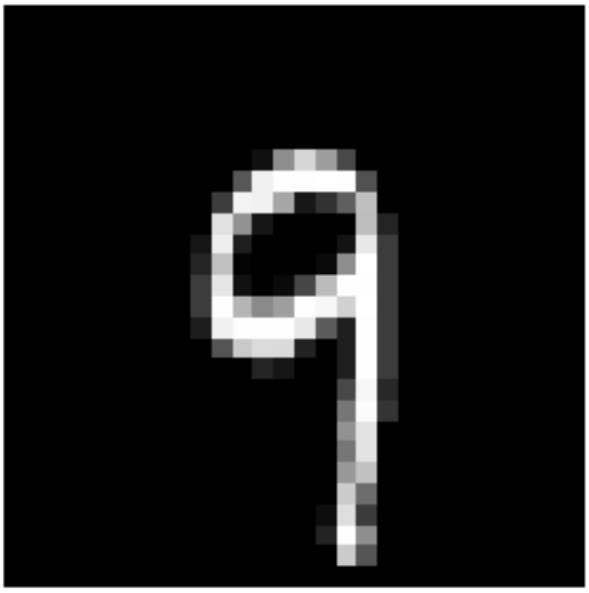
```
Label 9
A failure case:
1-NN classification:  9
True label:  3
The test image:
```

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz



Label 3

The corresponding nearest neighbor image:



Label 9

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

Task 4:

Computer: ThinkPad P40 Yoda

Specs: Intel Core i7-6600U (2.6 GHz base frequency, 3.4 GHz turbo frequency)

```
100% (500 of 500) |######################| Elapsed Time: 0:00:14 Time:  0:00:
14
Error of nearest neighbor classifier with Euclidean distance:  0.074
Classification time (seconds) with Euclidean distance:  14.423253059387207
```

Task 5:

```
## Computes Manhattan distance between two vectors.
def manh_dist(x,y):
    # input:
    # x, y: vectorization of an image of size 28 by 28
    # output:
    # the distance between the two vectors

    ### STUDENT: YOUR CODE HERE
    #sum = 0
    #for i in range(len(x)):
        #sum += ( abs(x[i] - y[i]) )
    sum = np.sum ( np.abs( np.array(x) - np.array(y) ) )
    return sum
    ### CODE ENDS

pbar = ProgressBar() # to show progress
## Predict on each test data point (and time it!)
t_before = time.time()
test_predictions = np.zeros(len(test_labels))
for i in pbar(range(len(test_labels))):
    test_predictions[i] = KNN_classifier(test_data[i,],train_data,train_labels,3,manh_dist)
```

```
t_after = time.time()


## Compute the error

err_positions = np.not_equal(test_predictions, test_labels)

error = float(np.sum(err_positions))/len(test_labels)


print("Error of nearest neighbor classifier with Manhattan distance: ", error)

print("Classification time (seconds) with Manhattan distance: ", t_after - t_before)
```

```
100% (500 of 500) |######################| Elapsed Time: 0:00:15 Time:  0:00:
15
Error of nearest neighbor classifier with Manhattan distance:  0.084
Classification time (seconds) with Manhattan distance:  15.655949354171753
```

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

Task 6:

```
 ## Compute a distance metric of your design
def my_dist(x,y):
    # input:
    # x, y: vectorization of an image of size 28 by 28
    # output:
    # the distance between the two vectors

    ### STUDENT: YOUR CODE HERE
    distance = np.sqrt( np.sum( ( np.array(x) - np.array(y) )**4 ) )
    return distance
    ### CODE ENDS

pbar = ProgressBar() # to show progress
## Predict on each test data point (and time it!)
t_before = time.time()
test_predictions = np.zeros(len(test_labels))
for i in pbar(range(len(test_labels))):
    test_predictions[i] = KNN_classifier(test_data[i,],train_data,train_la
bels,3,my_dist)

t_after = time.time()

## Compute the error
err_positions = np.not_equal(test_predictions, test_labels)
error = float(np.sum(err_positions))/len(test_labels)

print("Error of nearest neighbor classifier with the new distance: ", erro
r)
print("Classification time (seconds) with the new distance: ", t_after - t
_before)
```

```
100% (500 of 500) |######################| Elapsed Time: 0:00:25 Time:  0:00:
25
Error of nearest neighbor classifier with the new distance:  0.066
Classification time (seconds) with the new distance:  25.809839963912964
```

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

Task 7:

```
### STUDENT: YOUR CODE HERE

from matplotlib import pyplot as plt


def main():

    folds = 5


    #To be used for the plot#######################################

    Values_of_K = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] #X-axis

    Ks_Averaged_Error = [] #Y-axis

    Test_Error = [] #Y-axis

    ################################################################




    K_Error = list()

    Iteration_Error = list()

    folded_data = list()

    folded_labels = list()


    my_train_data = list()

    my_train_labels = list()

    my_test_data = list()

    my_test_labels = list()


    folded_data, folded_labels = cross_validation(train_data, train_labels, folds)



    #Loop for each value of K

    for KNN_K in range(1,11):
```

```
#Loop test for each fold while alternating the test_data and train_data and their respective labels

############################################################
# ITERATION 1 ##############################################
############################################################
my_test_data = folded_data[0]
my_test_labels = folded_labels[0]

my_train_data_1 = folded_data[1]
my_train_labels_1= folded_labels[1]

my_train_data_2 = folded_data[2]
my_train_labels_2= folded_labels[2]

my_train_data_3 = folded_data[3]
my_train_labels_3= folded_labels[3]

my_train_data_4 = folded_data[4]
my_train_labels_4 = folded_labels[4]

############################################################

## Predict on each test data point (and time it!)
pbar = ProgressBar() # to show progress
test_predictions = np.zeros(len(my_test_labels))
for i in pbar(range(len(my_test_labels))):
   test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_1, my_train_labels_1,
KNN_K, eucl_dist )
```

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

```
    ## Compute the error 1
    err_positions = np.not_equal(test_predictions, my_test_labels)
    error = float(np.sum(err_positions))/len(my_test_labels)

    K_Error.append(error)


    ## Predict on each test data point (and time it!)
    pbar = ProgressBar() # to show progress
    test_predictions = np.zeros(len(my_test_labels))
    for i in pbar(range(len(my_test_labels))):
       test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_2, my_train_labels_2,
KNN_K, eucl_dist )

    ## Compute the error 2
    err_positions = np.not_equal(test_predictions, my_test_labels)
    error = float(np.sum(err_positions))/len(my_test_labels)

    K_Error.append(error)


    ## Predict on each test data point (and time it!)
    pbar = ProgressBar() # to show progress
    test_predictions = np.zeros(len(my_test_labels))
    for i in pbar(range(len(my_test_labels))):
       test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_3, my_train_labels_3,
KNN_K, eucl_dist )
```

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

```
    ## Compute the error 3
    err_positions = np.not_equal(test_predictions, my_test_labels)
    error = float(np.sum(err_positions))/len(my_test_labels)

    K_Error.append(error)


    ## Predict on each test data point (and time it!)
    pbar = ProgressBar() # to show progress
    test_predictions = np.zeros(len(my_test_labels))
    for i in pbar(range(len(my_test_labels))):
       test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_4, my_train_labels_4,
KNN_K, eucl_dist )

    ## Compute the error 4
    err_positions = np.not_equal(test_predictions, my_test_labels)
    error = float(np.sum(err_positions))/len(my_test_labels)

    K_Error.append(error)



    ###############################################################
    # ITERATION 2 #################################################
    ###############################################################
    my_test_data = folded_data[1]
    my_test_labels = folded_labels[1]

    my_train_data_1 = folded_data[0]
```

```
    my_train_labels_1= folded_labels[0]


    my_train_data_2 = folded_data[2]

    my_train_labels_2 = folded_labels[2]


    my_train_data_3 = folded_data[3]

    my_train_labels_3 = folded_labels[3]


    my_train_data_4 = folded_data[4]

    my_train_labels_4 = folded_labels[4]


    ################################################################


     ## Predict on each test data point (and time it!)

    pbar = ProgressBar() # to show progress

    test_predictions = np.zeros(len(my_test_labels))

    for i in pbar(range(len(my_test_labels))):

       test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_1, my_train_labels_1,
KNN_K, eucl_dist )


    ## Compute the error 1

    err_positions = np.not_equal(test_predictions, my_test_labels)

    error = float(np.sum(err_positions))/len(my_test_labels)


    K_Error.append(error)



    ## Predict on each test data point (and time it!)

    pbar = ProgressBar() # to show progress
```

```
    test_predictions = np.zeros(len(my_test_labels))
    for i in pbar(range(len(my_test_labels))):
        test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_2, my_train_labels_2,
KNN_K, eucl_dist )

    ## Compute the error 2
    err_positions = np.not_equal(test_predictions, my_test_labels)
    error = float(np.sum(err_positions))/len(my_test_labels)

    K_Error.append(error)

    ## Predict on each test data point (and time it!)
    pbar = ProgressBar() # to show progress
    test_predictions = np.zeros(len(my_test_labels))
    for i in pbar(range(len(my_test_labels))):
        test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_3, my_train_labels_3,
KNN_K, eucl_dist )

    ## Compute the error 3
    err_positions = np.not_equal(test_predictions, my_test_labels)
    error = float(np.sum(err_positions))/len(my_test_labels)

    K_Error.append(error)

    ## Predict on each test data point (and time it!)
    pbar = ProgressBar() # to show progress
    test_predictions = np.zeros(len(my_test_labels))
```

```
    for i in pbar(range(len(my_test_labels))):
        test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_4, my_train_labels_4, KNN_K, eucl_dist )

    ## Compute the error 4
    err_positions = np.not_equal(test_predictions, my_test_labels)
    error = float(np.sum(err_positions))/len(my_test_labels)

    K_Error.append(error)



    ###############################################################
    # ITERATION 3 ################################################
    ###############################################################
    my_test_data = folded_data[2]
    my_test_labels = folded_labels[2]

    my_train_data_1 = folded_data[0]
    my_train_labels_1= folded_labels[0]

    my_train_data_2 = folded_data[1]
    my_train_labels_2 = folded_labels[1]

    my_train_data_3 = folded_data[3]
    my_train_labels_3 = folded_labels[3]

    my_train_data_4 = folded_data[4]
    my_train_labels_4 = folded_labels[4]
```

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

```
##############################################################


 ## Predict on each test data point (and time it!)
pbar = ProgressBar() # to show progress
test_predictions = np.zeros(len(my_test_labels))
for i in pbar(range(len(my_test_labels))):
    test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_1, my_train_labels_1,
KNN_K, eucl_dist )

## Compute the error 1
err_positions = np.not_equal(test_predictions, my_test_labels)
error = float(np.sum(err_positions))/len(my_test_labels)

K_Error.append(error)



## Predict on each test data point (and time it!)
pbar = ProgressBar() # to show progress
test_predictions = np.zeros(len(my_test_labels))
for i in pbar(range(len(my_test_labels))):
    test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_2, my_train_labels_2,
KNN_K, eucl_dist )

## Compute the error 2
err_positions = np.not_equal(test_predictions, my_test_labels)
error = float(np.sum(err_positions))/len(my_test_labels)

K_Error.append(error)
```

```
## Predict on each test data point (and time it!)
pbar = ProgressBar() # to show progress
test_predictions = np.zeros(len(my_test_labels))
for i in pbar(range(len(my_test_labels))):
    test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_3, my_train_labels_3, KNN_K, eucl_dist )

## Compute the error 3
err_positions = np.not_equal(test_predictions, my_test_labels)
error = float(np.sum(err_positions))/len(my_test_labels)

K_Error.append(error)


## Predict on each test data point (and time it!)
pbar = ProgressBar() # to show progress
test_predictions = np.zeros(len(my_test_labels))
for i in pbar(range(len(my_test_labels))):
    test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_4, my_train_labels_4, KNN_K, eucl_dist )

## Compute the error 4
err_positions = np.not_equal(test_predictions, my_test_labels)
error = float(np.sum(err_positions))/len(my_test_labels)

K_Error.append(error)
```

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

```
############################################################
# ITERATION 4 ##############################################
############################################################
my_test_data = folded_data[3]
my_test_labels = folded_labels[3]

my_train_data_1 = folded_data[0]
my_train_labels_1= folded_labels[0]

my_train_data_2 = folded_data[1]
my_train_labels_2 = folded_labels[1]

my_train_data_3 = folded_data[2]
my_train_labels_3 = folded_labels[2]

my_train_data_4 = folded_data[4]
my_train_labels_4 = folded_labels[4]

############################################################

 ## Predict on each test data point (and time it!)
pbar = ProgressBar() # to show progress
test_predictions = np.zeros(len(my_test_labels))
for i in pbar(range(len(my_test_labels))):
    test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_1, my_train_labels_1,
KNN_K, eucl_dist )
```

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

```
## Compute the error 1

err_positions = np.not_equal(test_predictions, my_test_labels)

error = float(np.sum(err_positions))/len(my_test_labels)


K_Error.append(error)



## Predict on each test data point (and time it!)

pbar = ProgressBar() # to show progress

test_predictions = np.zeros(len(my_test_labels))

for i in pbar(range(len(my_test_labels))):

   test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_2, my_train_labels_2,
KNN_K, eucl_dist )


## Compute the error 2

err_positions = np.not_equal(test_predictions, my_test_labels)

error = float(np.sum(err_positions))/len(my_test_labels)


K_Error.append(error)



## Predict on each test data point (and time it!)

pbar = ProgressBar() # to show progress

test_predictions = np.zeros(len(my_test_labels))

for i in pbar(range(len(my_test_labels))):

   test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_3, my_train_labels_3,
KNN_K, eucl_dist )


## Compute the error 3
```

```
    err_positions = np.not_equal(test_predictions, my_test_labels)
    error = float(np.sum(err_positions))/len(my_test_labels)

    K_Error.append(error)


    ## Predict on each test data point (and time it!)
    pbar = ProgressBar() # to show progress
    test_predictions = np.zeros(len(my_test_labels))
    for i in pbar(range(len(my_test_labels))):
       test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_4, my_train_labels_4,
KNN_K, eucl_dist )

    ## Compute the error 4
    err_positions = np.not_equal(test_predictions, my_test_labels)
    error = float(np.sum(err_positions))/len(my_test_labels)

    K_Error.append(error)



    ###############################################################
    # ITERATION 5 ################################################
    ###############################################################
    my_test_data = folded_data[4]
    my_test_labels = folded_labels[4]

    my_train_data_1 = folded_data[0]
    my_train_labels_1= folded_labels[0]
```

```
my_train_data_2 = folded_data[1]

my_train_labels_2 = folded_labels[1]


my_train_data_3 = folded_data[2]

my_train_labels_3 = folded_labels[2]


my_train_data_4 = folded_data[3]

my_train_labels_4 = folded_labels[3]


#################################################################


 ## Predict on each test data point (and time it!)

pbar = ProgressBar() # to show progress

test_predictions = np.zeros(len(my_test_labels))

for i in pbar(range(len(my_test_labels))):

   test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_1, my_train_labels_1,
KNN_K, eucl_dist )


## Compute the error 1

err_positions = np.not_equal(test_predictions, my_test_labels)

error = float(np.sum(err_positions))/len(my_test_labels)


K_Error.append(error)



## Predict on each test data point (and time it!)

pbar = ProgressBar() # to show progress

test_predictions = np.zeros(len(my_test_labels))
```

```
    for i in pbar(range(len(my_test_labels))):
        test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_2, my_train_labels_2, KNN_K, eucl_dist )

    ## Compute the error 2
    err_positions = np.not_equal(test_predictions, my_test_labels)
    error = float(np.sum(err_positions))/len(my_test_labels)

    K_Error.append(error)


    ## Predict on each test data point (and time it!)
    pbar = ProgressBar() # to show progress
    test_predictions = np.zeros(len(my_test_labels))
    for i in pbar(range(len(my_test_labels))):
        test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_3, my_train_labels_3, KNN_K, eucl_dist )

    ## Compute the error 3
    err_positions = np.not_equal(test_predictions, my_test_labels)
    error = float(np.sum(err_positions))/len(my_test_labels)

    K_Error.append(error)


    ## Predict on each test data point (and time it!)
    pbar = ProgressBar() # to show progress
    test_predictions = np.zeros(len(my_test_labels))
    for i in pbar(range(len(my_test_labels))):
```

```
        test_predictions[i] = KNN_classifier( my_test_data[i,], my_train_data_4, my_train_labels_4,
KNN_K, eucl_dist )


    ## Compute the error 4
    err_positions = np.not_equal(test_predictions, my_test_labels)
    error = float(np.sum(err_positions))/len(my_test_labels)


    K_Error.append(error)




    #Get the average error between all the errors calculated during the fold iterations
    Ks_Averaged_Error.append( Average_Error( K_Error ) )
    K_Error = []



#Do the same for test error##################################################
## Predict on each test data point (and time it!)

#Loop for each value of K
for KNN_K in range(1,11):


    pbar = ProgressBar() # to show progress
    t_before = time.time()
    test_predictions = np.zeros(len(test_labels))
    for i in pbar(range(len(test_labels))):
        test_predictions[i] = KNN_classifier(test_data[i,], train_data, train_labels, KNN_K, eucl_dist)
    t_after = time.time()
```

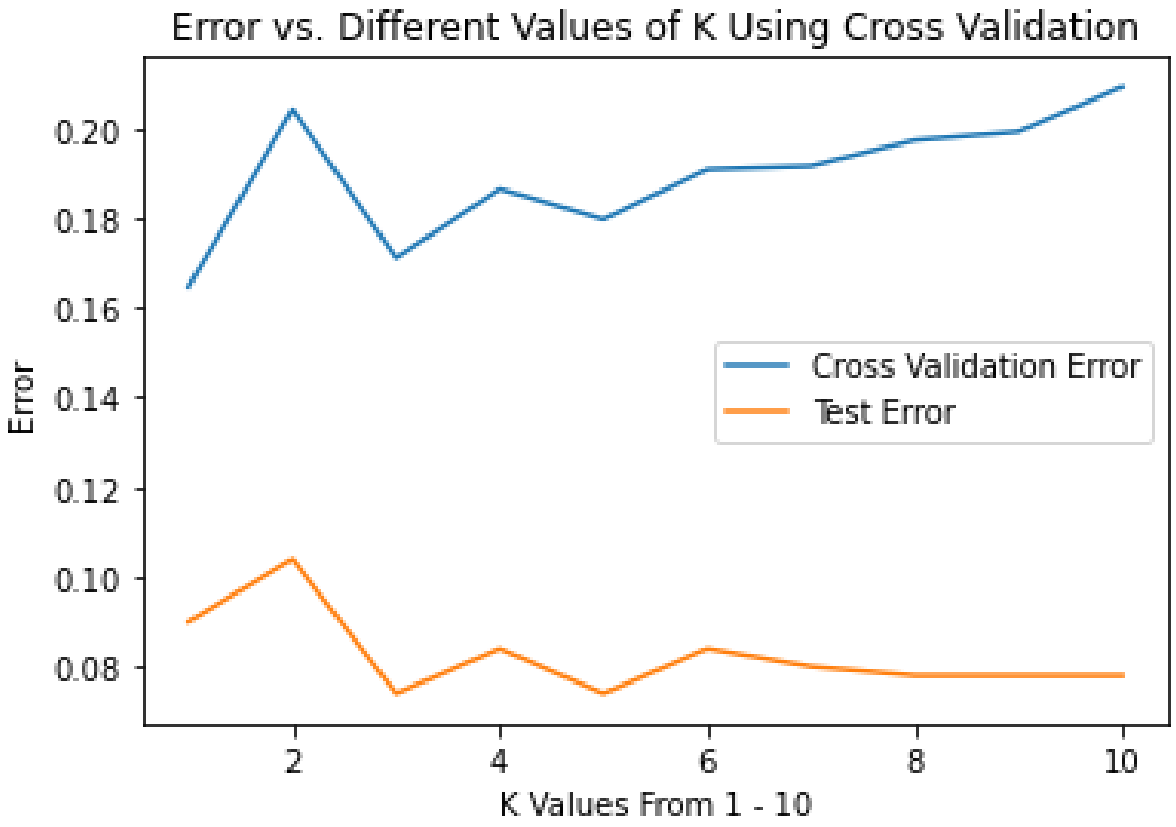
Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

```
        ## Compute the error
        err_positions = np.not_equal(test_predictions, test_labels)
        error = float(np.sum(err_positions))/len(test_labels)

        Test_Error.append(error)



    ##############################################################################
    ######## Now we plot the resutls ##############################################
    ##############################################################################

    #print("These were the results of 5-fold cross validation for K values 1-10:", Ks_Averaged_Error)
    plt.title("Error vs. Different Values of K Using Cross Validation")
    plt.xlabel("K Values From 1 - 10")
    plt.ylabel("Error")
    plt.plot(Values_of_K, Ks_Averaged_Error, label = "Cross Validation Error")
    plt.plot(Values_of_K, Test_Error, label = "Test Error")
    plt.legend()
    plt.show()



def cross_validation(dataset, labels, folds = 5):

    dataset_split = list()
    label_split = list()

#    dataset_copy = list(dataset)
#    label_copy  = list(labels)
```

```
    fold_size = int(len(dataset) / folds )

    dataset_1 = dataset[0: (fold_size)]
    dataset_2 = dataset[fold_size : (2*fold_size)]
    dataset_3 = dataset[(2*fold_size) : (3*fold_size)]
    dataset_4 = dataset[(3*fold_size) : (4*fold_size)]
    dataset_5 = dataset[(4*fold_size) : (5*fold_size)]

    labelset_1 = labels[0 : (fold_size)]
    labelset_2 = labels[fold_size : (2*fold_size)]
    labelset_3 = labels[(2*fold_size) : (3*fold_size)]
    labelset_4 = labels[(3*fold_size) : (4*fold_size)]
    labelset_5 = labels[(4*fold_size) : (5*fold_size)]

    dataset_split.append(dataset_1)
    dataset_split.append(dataset_2)
    dataset_split.append(dataset_3)
    dataset_split.append(dataset_4)
    dataset_split.append(dataset_5)

    label_split.append(labelset_1)
    label_split.append(labelset_2)
    label_split.append(labelset_3)
    label_split.append(labelset_4)
    label_split.append(labelset_5)

    return dataset_split, label_split
```

Collaborators: Alex Ryan, Bill Norris, and Ryan Stankiewicz

```
def Average_Error(List):
    return sum(List) / len(List)


if __name__=="__main__":
    main()
```



Based on the results I would probably select a K value of 3 since it's the lowest error result in the test error and second lowest cross validation error. There is also a noticeable pattern in both errors. At every even value for K, the error jumps, while at every odd value of K, the error decreases. This is probably due to the fact that it's harder to have a "majority vote" with an even number of participants.