Deniz Aytemiz
denizaytemiz@vt.edu
906470552

ECE 5424/CS5824 - Advanced Machine Learning - Problem Set 1

Question 1 - Supervised Learning

- The dataset used for this problem can be customers demographic informations such as income level, education level and location. In addition to that the behavioral patterns of a particular customer is important such as how many days they are using the streaming service in a week, how many hours or minutes they use it in a day, how much they stop the videos or switch the content without finishing it. Their subscription plan is also a good indicator since a monthly subscription or free trial is more likely to be cancelled then annual subcription users by intuitive guess. Therefore,
  X: customer age, customer location, customer subscription plan, number of days they use the service within a week, average amount of daily time of usage, number days passed since last usage of the service, number of unfinished content, number of stopping or switching the unfinished content.
  Y: label space can be 0/1 as an indicator of whether the customer is likely to cancel the subscription or not in the near future.
  L: cross-entropy loss, since it is used to minimize the predicted probability distribution and the actual probability distribution which is what the model is designed to achieve.
  H: is the all possible sets of decision trees used. The parameters of the decision trees, the maximum depth of the trees and the criteria of split.

  The problem is a binary classification problem and the model selected is random forest.

- The model can be used by the business in order to make a prediction on whether the customer is likely to cancel their subscription plan in the near future. By random forest different features will be randomly selected and trained in different decision trees which will outcome in either 0 or 1 as an estimate of the customer behaviour. Hence a predicition of the customer churn can be calculated. Random forest is selected since dataset contains many features which are both numerical and categorical. The dependence among the features are not clear hence it will also give an insight on the dependence or independence of the features. The usage of multiple trees with different features sets are more insightful in that sense with respect to logistic regression for instance, since in logistic regression all the features will be included in the model with different parameters and the interrelations of the features would be less included in the outcome.

Question 2 - Convex Optimization

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

Question 2.1:

If A is positive semi definite, then all eigenvalues of A should be non negative.

$$|A-\lambda . I| = \left| \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right| = 0$$

$$\left| \begin{bmatrix} 1-\lambda & 2 \\ 2 & 1-\lambda \end{bmatrix} \right| = 0$$

$(1-\lambda)^2 - 4 = 0$ → characteristic equation

$(1-\lambda)^2 = 4$, $1-\lambda = 2$ or $1-\lambda = -2$

$\lambda_1 = -1$, $\lambda_2 = 3$

Since λ_1 is negative, A is not positive semi definite.

An example of a positive semi-definite matrix would be

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

Question 2.2: A convex function is a continuos function whose value at the midpoint of every interval in its domain does not exceed the arithmetic mean of its values at the ends of the interval.

More generally, $f(x)$ is convex on interval $[a,b]$ for any two points x_1 & x_2 in $[a,b]$ and any λ where $0<\lambda<1$,

$$f[\lambda x_1 + (1-\lambda)x_2] \leq \lambda f(x_1) + (1-\lambda) \cdot f(x_2)$$

A concave function $f(x)$ is concave on interval $[a,b]$ if for any points x_1 & x_2 in $[a,b]$ the function $-f(x)$ is convex on that interval.

Affine functions represent vector-valued functions of the form

$$f(x_1, \ldots, x_n) = A_1 x_1 + \ldots + A_n x_n + b$$

The coefficients can be scalars or dense or sparse matrices. The constant term is a scalar or a column vector.

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

Question 2.3 : $f_1(w) = ||w||^2 = w^T w$

$f_2(\varepsilon) = \varepsilon$

Hessian matrix of $f_1(w)$ where $\bar{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_n \end{bmatrix}$ should be positive semi-definite.

$$H \text{ of } f_1(w) = \begin{bmatrix} \frac{\partial^2 f_1}{\partial^2 w_0} & \frac{\partial^2 f_1}{\partial w_0 w_1} & \cdots & \frac{\partial^2 f_1}{\partial w_0 w_n} \\ \frac{\partial^2 f_1}{\partial w_1 w_0} & \frac{\partial^2 f_1}{\partial w_1^2} & \cdots & \frac{\partial f_1}{\partial w_1 w_n} \\ \vdots & & & \\ \frac{\partial^2 f_1}{\partial w_n w_0} & \frac{\partial^2 f_1}{\partial w_n w_1} & \cdots & \frac{\partial^2 f_1}{\partial w_n^2} \end{bmatrix}$$

$$\frac{\partial^2 f_1}{\partial w_0^2} = 2 \;, \quad \frac{\partial f_1}{\partial w_0 w_1} = 0, \; \ldots$$

for $\frac{\partial^2 f_1}{\partial w_i^2} = 2$ for $i = 0, \ldots, n$ & $\frac{\partial^2 f_1}{\partial w_i w_j} = 0$ for $i = 0, \ldots, n$ & $j = 0, \ldots, n$ $i \neq j$

$$\text{therefore } H \text{ of } f_1(w) = \begin{bmatrix} 2 & 0 & \cdots & 0 \\ 0 & 2 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & & & \\ 0 & 0 & \cdots & 2 \end{bmatrix} = 2 \cdot I$$

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

Testing whether H of $f_1(w)$ is semi definite positive

$$x = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix}$$

$$x^T . H . x = [x_0 \dots x_n] \underbrace{\begin{bmatrix} 2 & 0 & \dots & 0 \\ \vdots & & & \\ 0 & 0 & \dots & 2 \end{bmatrix}}_{2I} \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} = 2 . \sum_{i=1}^{n} (x_i)^2 \geq 0$$

for all x.

Therefore H is a positive semi-definite which means $f_1(w)$ is convex.

Now considering $f_2(\mathcal{E}) = \mathcal{E}$

if $f_2(\mathcal{E})$ is convex on interval I, then for any $\mathcal{E}_1, \mathcal{E}_2 \in I$

$$\underbrace{f_2\left(\frac{\mathcal{E}_1 + \mathcal{E}_2}{2}\right)} \leq \frac{f_2(\mathcal{E}_1) + f_2(\mathcal{E}_2)}{2}$$

$$\frac{\mathcal{E}_1 + \mathcal{E}_2}{2} \leq \frac{\mathcal{E}_1 + \mathcal{E}_2}{2}$$ $\Rightarrow$ since this inequality holds

we can say $f_2(\mathcal{E}) = \mathcal{E}$ is convex function.

Since sum of convex functions will be convex too,

we can say $\sum_{i=1}^{n} \mathcal{E}_i$ is also convex function.

Therefore, $\underbrace{\|w\|^2}_{h(w)} + \underbrace{\sum_{i=1}^{n} \mathcal{E}_i}_{g(\mathcal{E})}$ will be convex too since

we showed $h(w)$ & $g(\mathcal{E})$ are convex functions & their sum

will be convex too.

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

Question 2.4 : $\min \|w\|^2 + \sum_{i=1}^{n} \varepsilon_i$

$$\text{s.t} \quad y_i w^T x_i \geq 1 - \varepsilon_i, \quad i = 1, \dots, n$$
$$\varepsilon_i \geq 0, \quad i = 1, \dots, n$$

we should write the above as :

$$\min \; f_0(w, \varepsilon)$$
$$\text{s.t} \quad g_i \leq 0, \quad i = 1, \dots, m$$
$$t_i \leq 0, \quad i = 1, \dots, n$$

$$y_i w^T x_i - 1 + \varepsilon_i \geq 0 \rightarrow \underbrace{-y_i w^T x_i + 1 - \varepsilon_i}_{g_i} \leq 0$$

$$\varepsilon_i \geq 0 \rightarrow \underbrace{-\varepsilon_i}_{t_i} \leq 0$$

$$f(x, \alpha, \beta) = \underbrace{\|w\|^2 + \sum_{i=1}^{n} \varepsilon_i}_{f_0(w,\varepsilon)} + \sum_{i=1}^{n} \alpha_i \cdot (-y_i w^T x_i + 1 - \varepsilon_i) + \sum_{i=1}^{n} \beta_i \cdot (-\varepsilon_i)$$

$$= \|w\|^2 + \sum_{i=1}^{n} \varepsilon_i - \sum_{i=1}^{n} \alpha_i y_i w^T x_i + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i \varepsilon_i - \sum_{i=1}^{n} \beta_i \varepsilon_i$$

$$= \|w\|^2 + \sum_{i=1}^{n} (1 - \alpha_i - \beta_i) \cdot \varepsilon_i + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i y_i w^T x_i$$

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

Question 2.5:

Assume $\bar{x}$ is feasible,

$$\mathcal{L}(\bar{x},\alpha,\beta) = f_0(\bar{x}) + \sum_{i=1}^{m} \alpha_i \cdot g_i(\bar{x}) + \sum_{i=1}^{p} \beta_i \cdot h_i(\bar{x})$$

we know that if $\bar{x}$ is feasible $g_i(\bar{x}) \leq 0$ & $h_i(\bar{x}) = 0$

hence Lagrangian reduces to:

$$\mathcal{L}(\bar{x},\alpha,\beta) = f_0(\bar{x}) + \underbrace{\sum_{i=1}^{m} \underbrace{\alpha_i}_{\geq 0} \cdot \underbrace{g_i(\bar{x})}_{\leq 0}}_{\text{this term is non-positive here}}$$

this term is non-positive here
it can be at maximum 0,
or it's gonna be negative.

Therefore $\max(\mathcal{L}(\bar{x},\alpha,\beta)) = f_0(\bar{x})$ where $\sum_{i=1}^{m} \alpha_i \cdot g_i = 0$

for any feasible $\bar{x}$.

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

Question 2.6 :

$$\max(\mathcal{L}(x,\alpha,\beta)) = \max\left(f_0(x) + \sum_{i=1}^{m} \alpha_i \cdot g_i(x) + \sum_{i=1}^{p} \beta_i \cdot h_i(x)\right)$$

we know that $f_0(x)$ & $g_i(x)$ are convex

$\alpha_i \cdot g_i(x)$ where $\alpha \geq 0$ will also be convex

and $\sum_{i=1}^{m} \alpha_i \cdot g_i(x)$, therefore will be convex too.

for a feasible $\bar{x}$ the max of Lagrangian is written as:

$$\max(\mathcal{L}(\bar{x},\alpha,\beta)) = \max\left(f_0(\bar{x}) + \sum_{i=1}^{m} \alpha_i \cdot g_i(\bar{x})\right)$$

where $f_0(\bar{x}) + \sum_{i=1}^{m} \alpha_i \cdot g_i(\bar{x})$ is convex.

from pointwise maximum property of convex functions, we can say $\max(\mathcal{L}(\bar{x},\alpha,\beta))$ is also convex since $g = \max\{f_1, \ldots, f_n\}$ is convex for all convex f_i functions.

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

Question 2.7 : assuming $\bar{x}$ is feasible

$$f(\bar{x}, \lambda, \nu) \leq f_0(\bar{x})$$

$$\min_x \{f(\bar{x}, \lambda, \nu)\} \leq f(\bar{x}, \lambda, \nu) \leq \underbrace{f_0(\bar{x})}_{\max(f(\bar{x}, \lambda, \nu))}$$

we know for feasible $\bar{x}$

$$\theta_p(\bar{x}) = f_0(\bar{x}) \quad \text{and} \quad \min_x f(x, \lambda, \nu) = \theta_D(\lambda, \nu)$$

therefore

$$\rightarrow \theta_D(\lambda, \nu) \leq f(\bar{x}, \lambda, \nu) \leq \theta_p(\bar{x})$$

Question 2.7:

$$\theta_p(x^*) = \max_{\alpha \geq 0, \beta} f(x^*, \alpha, \beta) = f_0(x^*)$$

$$\theta_D(\alpha^*, \beta^*) = \min_x f(x, \alpha^*, \beta^*)$$

for any α, β pairs & for $\bar{x}$ is feasible:

$$\min_x \{f(\bar{x}, \alpha, \beta)\} \leq f(\bar{x}, \alpha, \beta) \leq \underbrace{f_0(\bar{x})}_{= \theta_p(\bar{x})}$$

here $\min_x f(\bar{x}, \alpha^*, \beta^*) \leq \theta_p(\bar{x})$

if x^* is optimal, it is also feasible. so we can insert x^* instead of $\bar{x}$.

$$\min_x f(x^*, \alpha^*, \beta^*) \leq \theta_p(x^*)$$

$$= \theta_D(\alpha^*, \beta^*) \leq \theta_p(x^*)$$

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

Question 3 - KNN Programming Assignments

Task 1:

- Complete the code section to calculate the Euclidean distance. Copy the corresponding code here.

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

    ### STUDENT: YOUR CODE HERE
    return np.linalg.norm(x-y)
    ### CODE ENDS
```

- Testing the eucl_dist function:

```
index = random.sample(range(len(train_labels)), 10)
print(index)
for i in range(5):
    # Image index
    k = i * 2
    print("Distance from "+str(train_labels[index[k]])+" to "+str(train_labels[index[k+1]])+": "+
          str(eucl_dist(train_data[index[k],],train_data[index[k+1],])))
```

```
[1927, 317, 1371, 1093, 1024, 451, 1616, 334, 1802, 1832]
Distance from 0 to 8: 2578.628
Distance from 6 to 8: 2350.4702
Distance from 9 to 3: 2472.2693
Distance from 1 to 4: 2438.8745
Distance from 8 to 0: 2844.6978
```

Task 2:

- Complete the code sections for find KNN and KNN classifier. Copy the corresponding code here.

```
# Take a vector x and returns the indices of its K nearest neighbors
in the training set: train_data
def find_KNN(x, train_data, train_labels, K, dist):
    # Input:
    # x: test point
    # train_data: training data X
    # train_labels: training data labels y
```

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

```
    # 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 #####
    distance = np.zeros(len(train_data))
    # for each train_data point indexed at i,
    # take thes the euclidian distance and array dist stores it in
index i
    #dist = [||tr[0]-x|| ||tr[1]-x|| ||tr[2]-x|| ,.., ||tr[1999]-x||]
    for i in range(len(train_data)):
        distance[i] = dist(train_data[i],x)
    #then the elements of dist are sorted in ascending way and the
indexes of the first K elements are stored in m
    m = np.argsort(distance)[:K]
    return m
    ##### END OF CODE #####
```

```
# KNN classification
def KNN_classifier(x, train_data, train_labels,K,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 #####
    # m is the array of the indexes of training data points which
have the minimum distance with test point x,(in asceding order and K
of them)
    m = find_KNN(x, train_data, train_labels,K, dist)
    #if K=2, t[0]= train_labels[m[0]]=the index of training data
which has the min distance with x and t[1]=train_labels[m[1]]
```

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

```
    # so t[0] and t[1] will have different/same labels which may
range from 0 to 9
    t = np.zeros(K)
    for i in range(K):
        t[i] = train_labels[m[i]]
    n = np.round(t).astype(np.int64)
    #count holds number of occurances of elements ranging from 0 to
max(t)
    count = np.bincount(n)
    #mod returns the element which has the most occurance = mode of
the dataset = label of the data point
    mod = np.argmax(count)
    return mod
    ##### END OF CODE #####
```

Task 3:

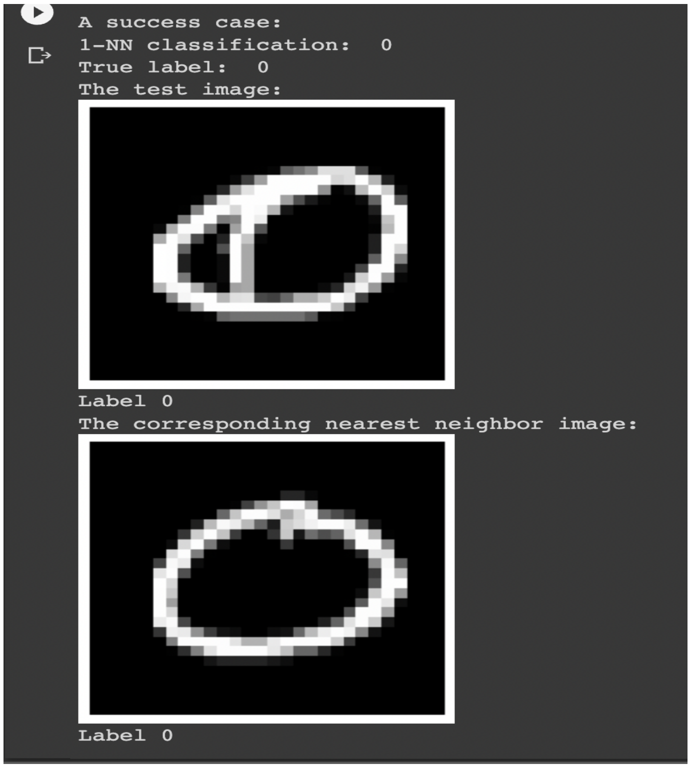
- Find one example of success case and one example of failed case for 1-nearest neighbor (i.e., K = 1). Print the outputs of the code and copy them here.

```
## A success case:
ind_success = 0  ### STUDENT: put one index of a success case here

print("A success case:")
print("1-NN classification: ",
KNN_classifier(test_data[ind_success,],train_data,train_labels,1,dis
t = eucl_dist))
print("True label: ", test_labels[ind_success])
print("The test image:")
vis_image(ind_success, "test")
print("The corresponding nearest neighbor image:")
vis_image(find_KNN(test_data[ind_success,],train_data,train_labels,1
,eucl_dist)[0], "train")
```

- Output:

Deniz Aytemiz
denizaytemiz@vt.edu
906470552



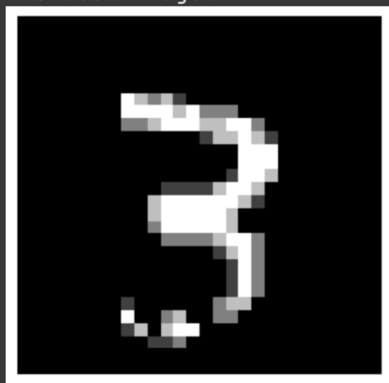

```
## A failure case:
ind_fail = 24  ### STUDENT: put one index of a success case here

print("A failed case:")
print("1-NN classification: ",
KNN_classifier(test_data[ind_fail,],train_data,train_labels,1,eucl_d
ist))
print("True label: ", test_labels[ind_fail])
print("The test image:")
vis_image(ind_fail, "test")
print("The corresponding nearest neighbor image:")
vis_image(find_KNN(test_data[ind_fail,],train_data,train_labels,1,eu
cl_dist)[0], "train")
```

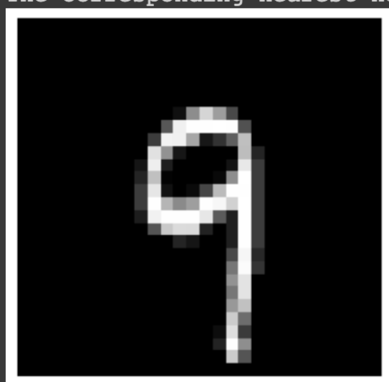
- Output:

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

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



```
Label 3
The corresponding nearest neighbor image:
```



```
Label 9
```

Task 4:

- What is the error of 3-nearest neighbor classifier with Euclidean distance? How long does it take? (also report the specs of the computer used to run the program)

Specs of computer:
Apple M1 chip
8 core CPU with 4 performance cores and 4 efficiency cores
7 core GPU
16 core Neutral Engine
8 GB unified memory
256 GB SSD

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

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

```
for i in pbar(range(len(test_labels))):
   test_predictions[i] =
KNN_classifier(test_data[i,],train_data,train_labels,3,eucl_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 Euclidean distance: ",
error)
print("Classification time (seconds) with Euclidean distance: ", t_after -
t_before)
#error_no=0
#for i in range(len(test_predictions)):
#     if test_predictions[i] != test_labels[i]:
#         error_no = error_no + 1
#         print(i)
```

- Output

```
100% (500 of 500) |######################| Elapsed Time: 0:00:09 Time:  0:00:09
Error of nearest neighbor classifier with Euclidean distance:  0.076
Classification time (seconds) with Euclidean distance:  9.3635995388031
```

Task 5:

- Complete the definition of manh dist and copy the code here. What is the error of 3-nearest neighbor classifier with Manhattan distance? How long does it take?

```
## 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
   dist = np.sum(np.abs(np.array(x)-np.array(y)))
   return dist
   ### CODE ENDS
```

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

```
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,dist =
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)
#error_no=0
#for i in range(len(test_predictions)):
#    if test_predictions[i] != test_labels[i]:
#        error_no = error_no + 1
#        print(i)
```

- Output:

```
100% (500 of 500) |######################| Elapsed Time: 0:00:11 Time:  0:00:11
Error of nearest neighbor classifier with Manhattan distance:  0.086
Classification time (seconds) with Manhattan distance:  11.675042867660522
```

Task 6:

- Define your own distance function and write down the mathematical definition. Copy the code here. What is the error of 3-nearest neighbor classifier with Manhattan distance? How long does it take? Note: you will only get full points if the self-defined distance function can improve over the Euclidean distance in terms of accuracy (worth 2 pts).

```
def minkowski_distance(x, y, p = 4):
   x = np.array(x)
   y = np.array(y)
```

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

```
    return np.power(np.sum(np.power(np.abs(x - y), p)), 1/p)
```

Mathematical definition of Minkowski distance is (in my code p = 4):

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

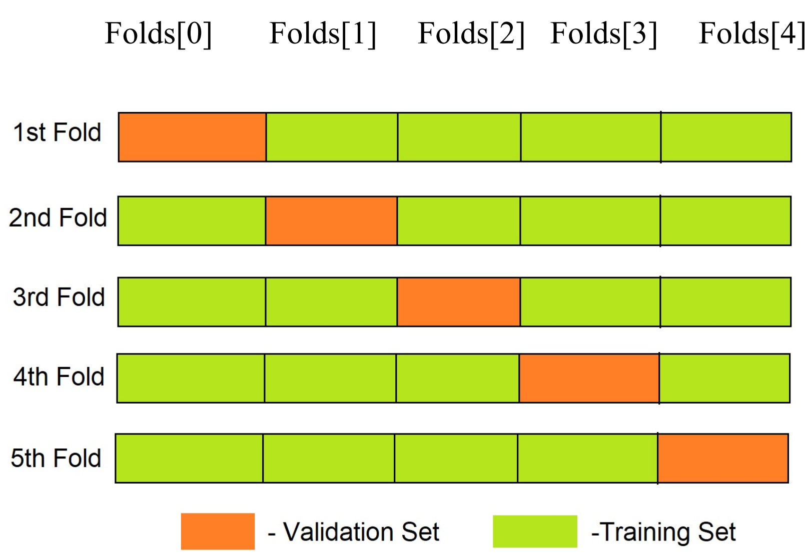
When p = 1 the distance metric is manhattan distance and when p = 2 it is euclidean distance.

- Output:

```
100% (500 of 500) |######################| Elapsed Time: 0:00:33 Time:  0:00:33
Error of nearest neighbor classifier with the new distance:  0.066
Classification time (seconds) with the new distance:  33.890695095062256
```

Task 7:

- Implement the 5-fold cross validation to choose the best K (number of nearest neighbors) between 1 and 10 for KNN with Euclidean distance. Copy the code to the solution file and plot the 5-fold validation error with respect to K. Also plot the test error on the same figure. Which K would you choose? What are some other observations you can make?



How I tackled the problem: For each iteration where the validation and training set changes, I have trained the model with k's ranging from 1 to 10 and saved the errors in an array for each k

Deniz Aytemiz
denizaytemiz@vt.edu
906470552
value and I have plotted them. After the 5th iteration, meaning every fold has been validation set, I take average of the errors hence I end up with an array in the form of [average 5-fold error for k = 1, average 5-fold error for k = 2, average 5-fold error for k = 3, average 5-fold error for k = 4, average 5-fold error for k = 5, average 5-fold error for k = 6, average 5-fold error for k = 7, average 5-fold error for k = 8, average 5-fold error for k = 9, average 5-fold error for k = 10]. Then I plotted that with the test error.

The code below is for folds[0] is validation set and folds[1], folds[2], folds[3], folds[4] are training. So, it is the first iteration of k-fold.

```
### STUDENT: YOUR CODE HERE

# index array is the indexes shuffled ranging from 0 to 1999
index_array = np.arange(0, len(train_data))
np.random.shuffle(index_array)
# these shuffled indices are divided into 5 equally sized segments
folds = np.array_split(index_array, 5)
# for this iteration folds[0] is validation set and the rest is
training set
#train_new_indices are the indices for new training set
train_new_indices =
np.concatenate((folds[1],folds[2],folds[3],folds[4]), axis = 0)

#new training set and training labels are created for the 5-fold
validation here
train_new = []
train_new_labels = []
for j in train_new_indices:
   train_new.append(train_data[j])
   train_new_labels.append(train_labels[j])

#error_array0 is gonna hold the error for k = 1 to k = 10,
#meaning error_array0[0] = error for k = 1 and error_array0[9] =
error for k = 10
error_array0=[]

# for different values of nearest neighbors, and folds[0] is the
validation set,
# the model is trained with new training set train_new consisting
from folds[1]...folds[4]
```

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

```
# and corresponding training labels which are train_new_labels, each
prediction is saved
# to test_predictions_new[]
for k in range(1,11):
 test_predictions_new = np.zeros(len(folds[0]))
 j = 0
 for i in folds[0]:
     x = train_data[i]
     test_predictions_new[j] =
KNN_classifier(x,train_new,train_new_labels,k,eucl_dist)
     j = j+1
     #print(i)

# error_no is the total number of points where the prediction is not
same with
# the actual label of the test point
 error_no=0
 for i in range(len(test_predictions_new)):
     if test_predictions_new[i] != train_labels[folds[0][i]]:
         error_no = error_no + 1
         #print(i)
# error_array0 holds the error for each K in KNN where folds[0] is
the validation set.
 print('k is ', k, 'error point is', error_no)
 error_array0.append(error_no/len(test_predictions_new))
```

- Output: the k's are KNN parameter k's and the error point is the total number of misclassified data points for that particular k.

```
k is  1 error point is 49
k is  2 error point is 62
k is  3 error point is 45
k is  4 error point is 46
k is  5 error point is 44
k is  6 error point is 49
k is  7 error point is 48
k is  8 error point is 50
k is  9 error point is 51
k is  10 error point is 59
```

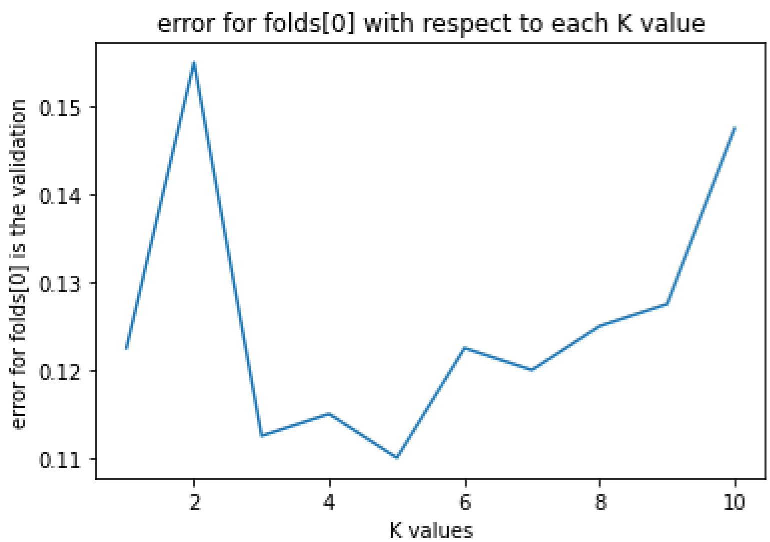
Deniz Aytemiz
denizaytemiz@vt.edu
906470552

To see the relation between k's and the corresponding error, I put them in a graph.(error is scaled between 0 -1 by dividing the total number of misclassified data point to length of prediction array.)

The code for this is:

```
import matplotlib.pyplot as plt
K = [1,2,3,4,5,6,7,8,9,10]
# plot the error with respect to number of K for folds[0] is the validation set
plt.plot(K, error_array0)
plt.xlabel('K values')
plt.ylabel('error for folds[0] is the validation')
plt.title('error for folds[0] with respect to each K value')
plt.show()
```

The output graph is:



Now for the second iteration where folds[1] is the validation and folds[0], folds[2], folds[3], folds[4] are the training set, the code is given as follows:

```
#fold [1] is validation set

train_new_indices =
np.concatenate((folds[0],folds[2],folds[3],folds[4]), axis = 0)
```

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

```
      train_new = []
      train_new_labels = []
      for j in train_new_indices:
         train_new.append(train_data[j])
         train_new_labels.append(train_labels[j])

      error_array1=[]


      for k in range(1,11):
       test_predictions_new = np.zeros(len(folds[1]))
       j = 0
       for i in folds[1]:
           x = train_data[i]
           test_predictions_new[j] =
KNN_classifier(x,train_new,train_new_labels,k,eucl_dist)
           j = j+1
           #print(i)


       error_no=0
       for i in range(len(test_predictions_new)):
           if test_predictions_new[i] != train_labels[folds[1][i]]:
               error_no = error_no + 1
               #print(i)
       # error_array1 holds the error for each K in KNN where folds[1] is
the validation set.

       print('k is =', k, 'error point is', error_no)
       error_array1.append(error_no/len(test_predictions_new))
```

The output: The k's are KNN parameter k's and the error point is the total number of misclassified data points for that particular k.

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

```
k is = 1 error point is 39
k is = 2 error point is 49
k is = 3 error point is 39
k is = 4 error point is 38
k is = 5 error point is 42
k is = 6 error point is 42
k is = 7 error point is 42
k is = 8 error point is 41
k is = 9 error point is 44
k is = 10 error point is 43
```

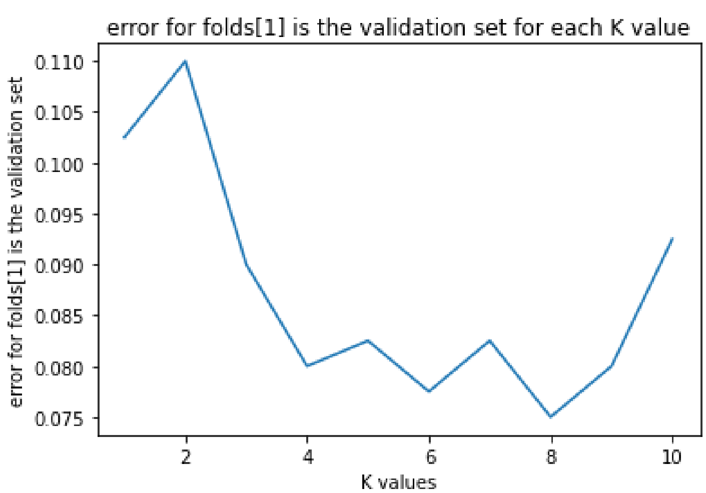
To see the relation between k's and the corresponding error, I put them in a graph.(error is scaled between 0 -1 by dividing the total number of misclassified data point to length of prediction array.)

The code for this is:

K is the same array defined before which is [1,2,3,4,5,6,7,8,9,10]

```
plt.plot(K, error_array1)
plt.xlabel('K values')
plt.ylabel('error for folds[1] is the validation set')
plt.title('error for folds[1] is the validation set for each K value')
plt.show()
```

The output graph:



Deniz Aytemiz
denizaytemiz@vt.edu
906470552

Now for the second iteration where folds[2] is the validation and folds[0], folds[1], folds[3], folds[4] are the training set, the code is given as follows:

```
      #folds[2] is validation set


      train_new_indices =
np.concatenate((folds[0],folds[1],folds[3],folds[4]), axis = 0)


      train_new = []
      train_new_labels = []
      for j in train_new_indices:
         train_new.append(train_data[j])
         train_new_labels.append(train_labels[j])

      error_array2=[]

      for k in range(1,11):
       test_predictions_new = np.zeros(len(folds[2]))
       j = 0
       for i in folds[2]:
           x = train_data[i]
           test_predictions_new[j] =
KNN_classifier(x,train_new,train_new_labels,k,eucl_dist)
           j = j+1
           #print(i)


       error_no=0
       for i in range(len(test_predictions_new)):
           if test_predictions_new[i] != train_labels[folds[2][i]]:
               error_no = error_no + 1
               #print(i)
       # error_array2 holds the error for each K in KNN where folds[2] is
the validation set.

       print('k is =', k, 'error point is', error_no)
       error_array2.append(error_no/len(test_predictions_new))
```

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

- The output:

```
k is = 1 error point is 50
k is = 2 error point is 60
k is = 3 error point is 44
k is = 4 error point is 42
k is = 5 error point is 39
k is = 6 error point is 37
k is = 7 error point is 39
k is = 8 error point is 39
k is = 9 error point is 38
k is = 10 error point is 39
```

To see the relation between k's and the corresponding error, I put them in a graph.(error is scaled between 0 -1 by dividing the total number of misclassified data point to length of prediction array.)

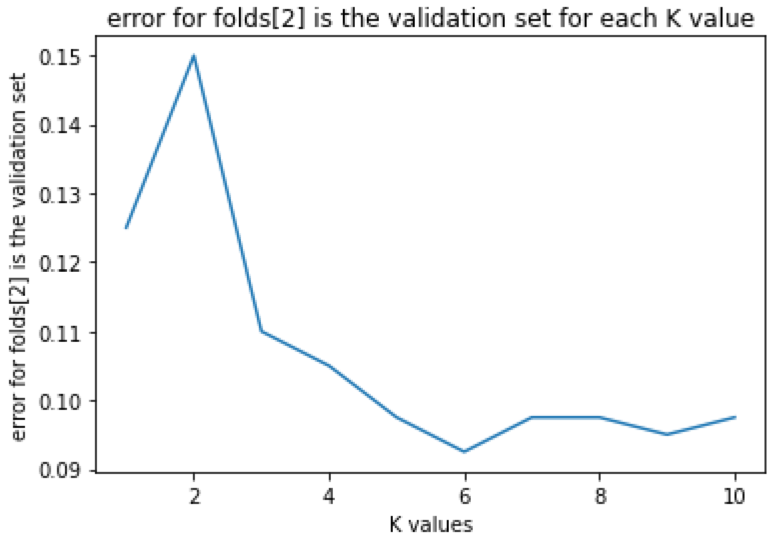
The code for this is:

K is the same array defined before which is [1,2,3,4,5,6,7,8,9,10]

```
    plt.plot(K, error_array2)
    plt.xlabel('K values')
    plt.ylabel('error for folds[2] is the validation set')
    plt.title('error for folds[2] is the validation set for each K
value')
    plt.show()
```

The output graph:

Deniz Aytemiz
denizaytemiz@vt.edu
906470552



Now for the second iteration where folds[3] is the validation and folds[0], folds[1], folds[2], folds[4] are the training set, the code is given as follows:

```
#folds[3] is validation set

train_new_indices = np.concatenate((folds[0],folds[1],folds[2],folds[4]),
axis = 0)


train_new = []
train_new_labels = []
for j in train_new_indices:
   train_new.append(train_data[j])
   train_new_labels.append(train_labels[j])

error_array3=[]

for k in range(1,11):
 test_predictions_new = np.zeros(len(folds[3]))
 j = 0
 for i in folds[3]:
     x = train_data[i]
     test_predictions_new[j] =
KNN_classifier(x,train_new,train_new_labels,k,eucl_dist)
     j = j+1
```

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

```
    #print(i)


error_no=0
for i in range(len(test_predictions_new)):
    if test_predictions_new[i] != train_labels[folds[3][i]]:
        error_no = error_no + 1
        #print(i)
  # error_array3 holds the error for each K in KNN where folds[3] is the
validation set.
print('k is =', k, 'error point is', error_no)
error_array3.append(error_no/len(test_predictions_new))
```

- The output:

```
k is = 1 error point is 41
k is = 2 error point is 49
k is = 3 error point is 41
k is = 4 error point is 35
k is = 5 error point is 37
k is = 6 error point is 41
k is = 7 error point is 37
k is = 8 error point is 43
k is = 9 error point is 37
k is = 10 error point is 38
```

To see the relation between k's and the corresponding error, I put them in a graph.(error is scaled between 0 -1 by dividing the total number of misclassified data point to length of prediction array.)

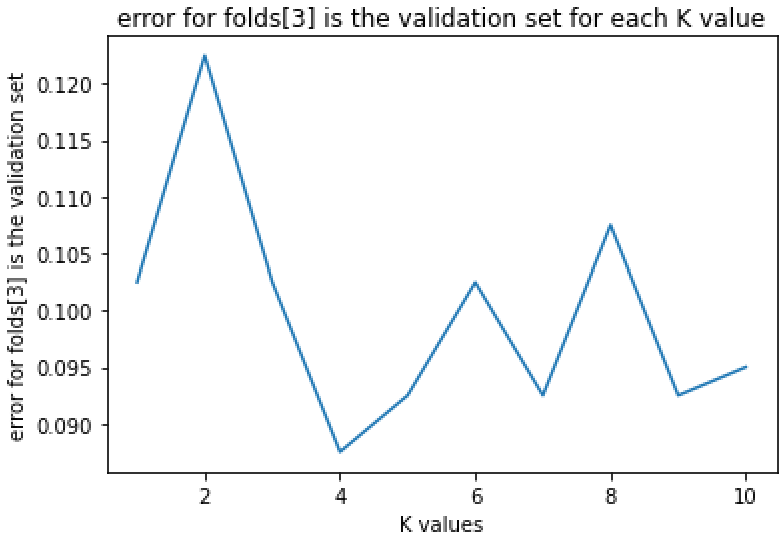
The code for this is:

K is the same array defined before which is [1,2,3,4,5,6,7,8,9,10]

```
plt.plot(K, error_array3)
plt.xlabel('K values')
plt.ylabel('error for folds[3] is the validation set')
plt.title('error for folds[3] is the validation set for each K value')
plt.show()
```

The output graph:

Deniz Aytemiz
denizaytemiz@vt.edu
906470552



Now for the second iteration where folds[4] is the validation and folds[0], folds[1], folds[2], folds[3] are the training set, the code is given as follows:

```
#folds[4] is validation set

train_new_indices = np.concatenate((folds[0],folds[1],folds[2],folds[3]),
axis = 0)


train_new = []
train_new_labels = []
for j in train_new_indices:
   train_new.append(train_data[j])
   train_new_labels.append(train_labels[j])

error_array4=[]

for k in range(1,11):
 test_predictions_new = np.zeros(len(folds[4]))
 j = 0
 for i in folds[4]:
     x = train_data[i]
     test_predictions_new[j] =
KNN_classifier(x,train_new,train_new_labels,k,eucl_dist)
     j = j+1
     #print(i)
```

Deniz Aytemiz
denizaytemiz@vt.edu
906470552

```
error_no=0
for i in range(len(test_predictions_new)):
    if test_predictions_new[i] != train_labels[folds[4][i]]:
        error_no = error_no + 1
        #print(i)
 # error_array4 holds the error for each K in KNN where folds[4] is the validation set.
print('k is =', k, 'error point is', error_no)
error_array4.append(error_no/len(test_predictions_new))
```

The output:

```
k is = 1 error point is 40
k is = 2 error point is 45
k is = 3 error point is 38
k is = 4 error point is 37
k is = 5 error point is 37
k is = 6 error point is 41
k is = 7 error point is 40
k is = 8 error point is 40
k is = 9 error point is 43
k is = 10 error point is 47
```

To see the relation between k's and the corresponding error, I put them in a graph.(error is scaled between 0 -1 by dividing the total number of misclassified data point to length of prediction array.)

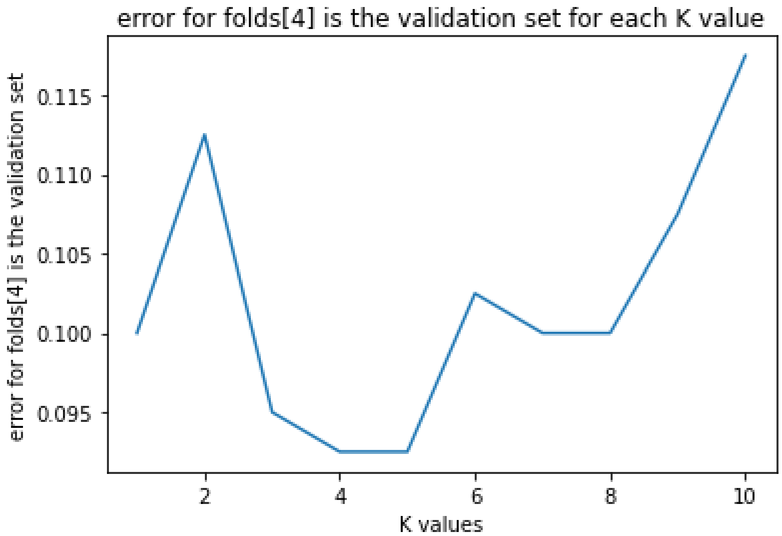
The code for this is:

K is the same array defined before which is [1,2,3,4,5,6,7,8,9,10]

```
plt.plot(K, error_array4)
plt.xlabel('K values')
plt.ylabel('error for folds[4] is the validation set')
plt.title('error for folds[4] is the validation set for each K value')
plt.show()
```

The output graph:

Deniz Aytemiz
denizaytemiz@vt.edu
906470552



Finally, the average of `error_array0, error_array1, error_array2, error_array3, error_array4` taken and it is plotted which is the average 5-folds error for each k.

The code for this is as follows:

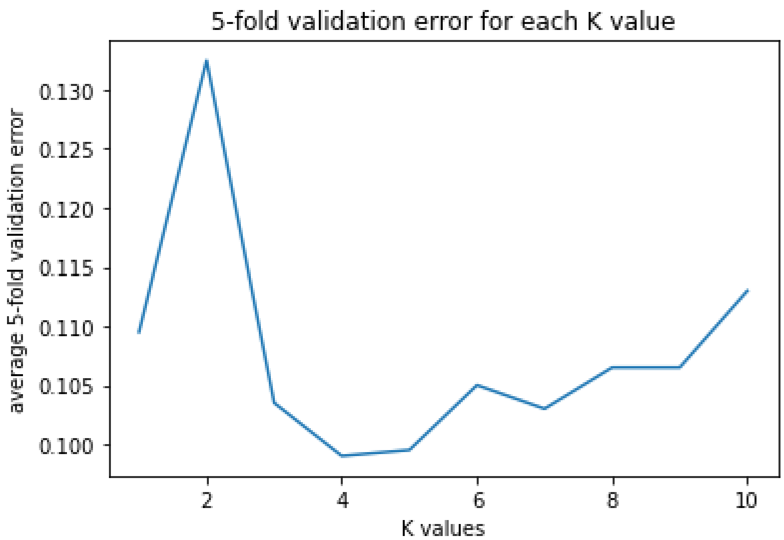
```
error_average=[]
for i in range(10):
 av= (error_array0[i] + error_array1[i] + error_array2[i] +
error_array3[i] + error_array4[i])/5
 error_average.append(av)
```

```
#import matplotlib.pyplot as plt


plt.plot(K, error_average)
plt.xlabel('K values')
plt.ylabel('average 5-fold validation error')
plt.title('5-fold validation error for each K value')
plt.show()
```

The output graph:

Deniz Aytemiz
denizaytemiz@vt.edu
906470552



Now the test set is used for testing the model and whole training set and k values ranging from ` to 10 are used for training the model. The code for this is as follows:

```
#test set is used for different k's

error_test = []
for k in range(1,11):
 test_predictions_new = np.zeros(len(test_labels))
 j = 0
 for i in test_data:
     test_predictions[j] =
KNN_classifier(i,train_data,train_labels,k,eucl_dist)
     j = j+1
     #print(i)

 error_no=0
 for i in range(len(test_predictions)):
     if test_predictions[i] != test_labels[i]:
         error_no = error_no + 1
         #print(i)
  print('k is =', k, 'error point is', error_no)
 error_test.append(error_no/len(test_predictions))
```

The output(the test error for each k):

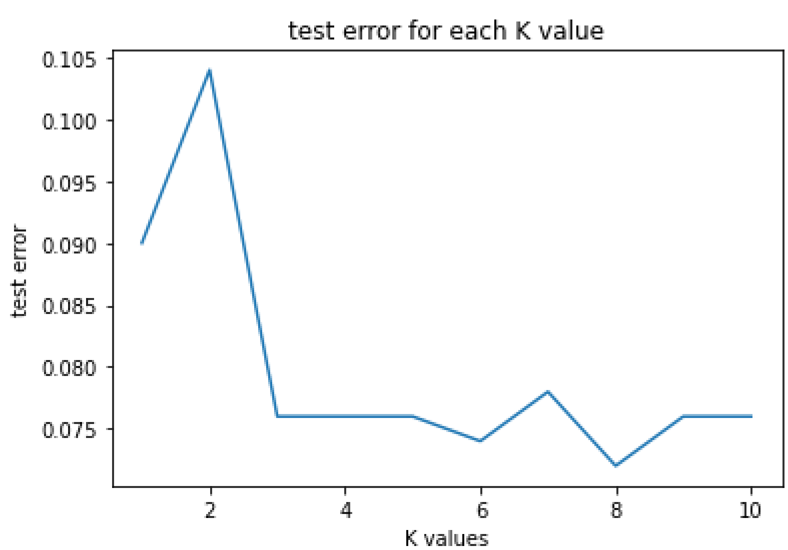
Deniz Aytemiz
denizaytemiz@vt.edu
906470552

```
k is = 1 error point is 45
k is = 2 error point is 52
k is = 3 error point is 38
k is = 4 error point is 38
k is = 5 error point is 38
k is = 6 error point is 37
k is = 7 error point is 39
k is = 8 error point is 36
k is = 9 error point is 38
k is = 10 error point is 38
```

For plotting the test error graph, the code is as follows:

```
#test error is plotted for each k
plt.plot(K, error_test)
plt.xlabel('K values')
plt.ylabel('test error')
plt.title('test error for each K value')
plt.show()
```

The output graph is as follows:



Now plotting both test error and average 5-folds error on the same graph, the code is as follows:

```
# plot the average 5 fold validation error and test error on same graph
fig, ax = plt.subplots()
ax.plot(K, error_average , label='average 5 fold validation error')
ax.plot(K, error_test , label='test error')
```

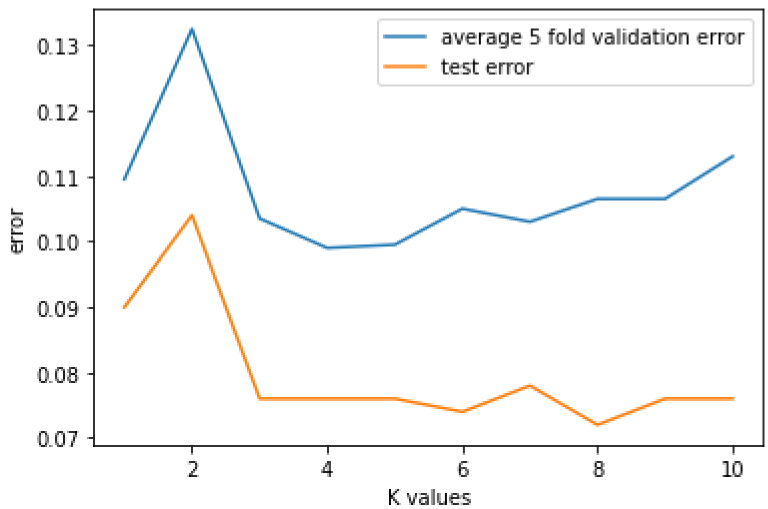
Deniz Aytemiz
denizaytemiz@vt.edu
906470552

```
ax.set_xlabel('K values')
ax.set_ylabel('error')

ax.legend()

plt.show()
```

The output graph:



Observations:
The K I would choose would be 4 since it gives very little error for both validation and test. What I have observed is that both graphs have similar patters. The error gets maximized when K = 2 and drastically decreases at K = 3. For K > 3 the error does not drastically changes but minimally differs. For K > 5 one can observe the validation error starts to increase which probably means the model is starting to get overfit.And after K = 9 the validation error increases more sharply. Also the test error is always smaller than the validation error.