906200395

906134545

906077496

1. In order to solve the problem of customer churn I would suggest using a machine learning algorithm that keeps track of movie ratings of movies that are watched, in order to improve suggestions. Every time the user watches a movie or while they watch the movie we can allow them to give a rating. This would tell the algorithm if a suggested movie was either a good or bad suggestion. We can include things like particular directors, actors and genres in the feature space. Data would be given from movies that a user has watched. The label space would be the suggested movie. The loss function should be an absolute loss function, we shouldn't focus too hard on the outliers, but rather allow the user to select the movie that they find enjoyable. We don't want our recommendations to be limiting in its scope either. Our Hypothesis space would be that given the movies actors, genre, director, and producer, we would use their past history in order to guess a movie they want to watch. This would be a classification problem. Using the input variables to create a classification of good suggestion, or bad suggestion.

2. Compare "machine learning for free" with Euclidian distance

ID	Money	Free	For	Gambling	Fun	Machine	Learning	Distance
1	3	0	0	0	0	0	0	Sqrt(13)
2	1	2	1	1	1	0	0	Sqrt(6)
3	0	0	1	1	1	0	0	Sqrt(5)
4	0	0	1	0	3	1	1	Sqrt(10)
5	0	1	0	0	0	1	1	Sqrt(1)

   a. The table of test data is compared to the phrase shown above. The Euclidian distance is shown in the column. To find the KNN where K is 1 for the lowest number. This would be ID 5 which is not spam. So we can assume that the test phrase is not spam.
   b. Using the same graph we now instead look for the lowest 3 values and take the average. These would correspond to ID 5,2,3. ID 2 and 3 are spam so this learning algorithm would assume it is spam

ID	Money	Free	For	Gambling	Fun	Machine	Learning	Distance
1	3	0	0	0	0	0	0	7
2	1	2	1	1	1	0	0	6
3	0	0	1	1	1	0	0	5
4	0	0	1	0	3	1	1	4
5	0	1	0	0	0	1	1	1

   c. Using the Manhattan distance comparing the table to the phrase "machine learning for free" we get the corresponding distances in the column distances as shown above. We look for the lowest 3 values and take the average. These would correspond to ID 5,4,3. ID 4 and 5 are not spam so this learning algorithm would assume it its not spam

ID	Money	Free	For	Gambling	Fun	Machine	Learning	Distance
1	3	0	0	0	0	0	0	0/(4*3)

2	1	2	1	1	1	0	0	3/(4*6)
3	0	0	1	1	1	0	0	1/(4*3)
4	0	0	1	0	3	1	1	3/(4*6)
5	0	1	0	0	0	1	1	3/(4*3)
T	0	1	1	0	0	1	1	---

d. Using the cosine similarity comparing the table to the phrase "machine learning for free" we get the corresponding distances in the column distances as shown above. We look for the lowest 3 values and take the average. These would correspond to ID 1,2,4. ID 1 and 2 are spam so this learning algorithm would assume it is spam

e. Consider the squared chord distance, calculated by $\sum_{i=1}^{n}\left(\sqrt{a_i}-\sqrt{b_i}\right)^2$. This formula highlights differences between a,b and minimizes similarities being compared with the Manhattan and Euclidian distance formulas

ID	Money	Free	For	Gambling	Fun	Machine	Learning	Distance
1	3	0	0	0	0	0	0	7
2	1	2	1	1	1	0	0	5.17
3	0	0	1	1	1	0	0	5
4	0	0	1	0	3	1	1	4
5	0	1	0	0	0	1	1	1
T	0	1	1	0	0	1	1	---

Using KNN of 3 we see that ID 4 is again the closest to the value 3 and still the item remains not spam.

3.

a.

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

- Universal set U has 36 possibilities
- Let E be the event that the sum of the dice is even. There are 18 even pairs of numbers. P(E) = 18/36=1/2.
- Let F be the event that at least one of the dice lands on 6. This happens in 11 cases. P(F) = 11/36
- Let G be the event that the numbers on the two dices are equal. This happens in 6 cases P(G) = 6/36=1/6
- P(E∪F)= = P(F)+P(G)- P(F∩G)=18/36+11/36-5/36 = 24/36 = 2/3
- P(E∩F)=P(F) that are even numbers. There are 11 numbers in the set, but only 5 are even. The answer is 5/36

- P(F∪G)=P(F)+P(G)- P(F∩G)= there is only one number that has at least one 6 but also where the two dice are equal(6,6). P(F)= P(F∩!G)=P(F-1)=10/36. P(F∪G)=10/36+6/36 = 16/36 = 4/9
- P(F∩G)=1/36

b.

	1	2	3	4	5	6
1	0	1	2	3	4	5
2	1	0	1	2	3	4
3		1	0	1		
4			1	0	1	
5				1	0	1
6					1	0

- Let P(D) be the probability that the difference of the dice rolled is by 1 = 10/36=5/18
- Let P(S) be the probity the dice are the same = 6/36 = 1/6
- Total outcomes T has 10+6=16 total outcome. P(T) = 16/36 = 4/9
- P(!T) = 1- 4/9 = 5/9 (the probability that we get nether a difference of one or two dice that are the same)
- If we roll n times, and the final roll the dice are the same, our probability would be equal to $P\big(E\ (n)\big) = (5/9)^{n-1} * (\frac{1}{6})$
- Because we don't know what the value of n is we need to take the sum of all possibilities for an infinite number of rolls
- $\mathrm{P(E)} = \sum_{n=1}^{\infty} (5/9)^{n-1} * (\frac{1}{6})$
  $= (\frac{1}{6})(\sum_{n=1}^{\infty}(5/9)^{n}$
  $= (\frac{1}{6})(1 + \sum_{n=1}^{\infty}(5/9)^{n}$ //we can do this because the first number in the set is one
  $= \frac{1}{6} * \left(1 + \frac{\frac{5}{9}}{1-\left(\frac{5}{9}\right)}\right) = \left(\frac{1}{6}\right)\left(\frac{9}{4}\right) = \frac{9}{24} = \frac{3}{8}$

c.

- Picking urn A P(A) = ½
- Picking urn B P(B) = ½
- Picking red r given A p(r|A) = 99/100
- Picking red given B P(r|B) = 1/100
- Picking red P(r) = P(r|A)P(A)+P(r|B)P(B)=(99/100*1/2)+(1/100*1/2)=1/2
- P(A|r) = P(r∩A)/p(r) = $= \frac{P(r|A)*P(A)}{P(r)} = \frac{\frac{99}{100}*\frac{1}{2}}{\frac{1}{2}} = \frac{99}{100}$

d.

- The matrix A is a m*n matrix. There are n columns describe by w(i) from 0 to m. these columns are al orthogonal as described.
- Since A has orthogonal columns, $A^T A$ is an n*n matrix with diagonal values corresponding to the eigenvalues σ(i) or the length of w. this n*n matrix of eigenvalues Σ is the m*n matrix of eigenvalues 0<i<n.
- because U Σ should be the eigen value array, we can conclude U is I(identity matrix size m*m)
- to maintain A = UΣV^T where UΣ is the eigen value array and A is an orthogonal array determine by w(i), we can calculate V^T to be the matrix where the ith contain w(i)/ σ(i)

4. Before I begin, I change the first block of code to be

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import time
import random
#!pip install ProgressBar
from progressbar import ProgressBar
import math
#!{sys.executable} -m pip install ProgressBar
#!pip install ProgressBar
from random import randrange
from copy import copy, deepcopy
```

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import time
import random
#!pip install ProgressBar
from progressbar import ProgressBar
import math
#!{sys.executable} -m pip install ProgressBar
#!pip install ProgressBar
from random import randrange
from copy import copy, deepcopy

a.

```
## 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

    d = len(x)
    sum = 0

    for i in range(d):
        sum = sum + pow((x[i]-y[i]),2)

    sum = math.sqrt(sum)
    return sum
### CODE ENDS
```

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

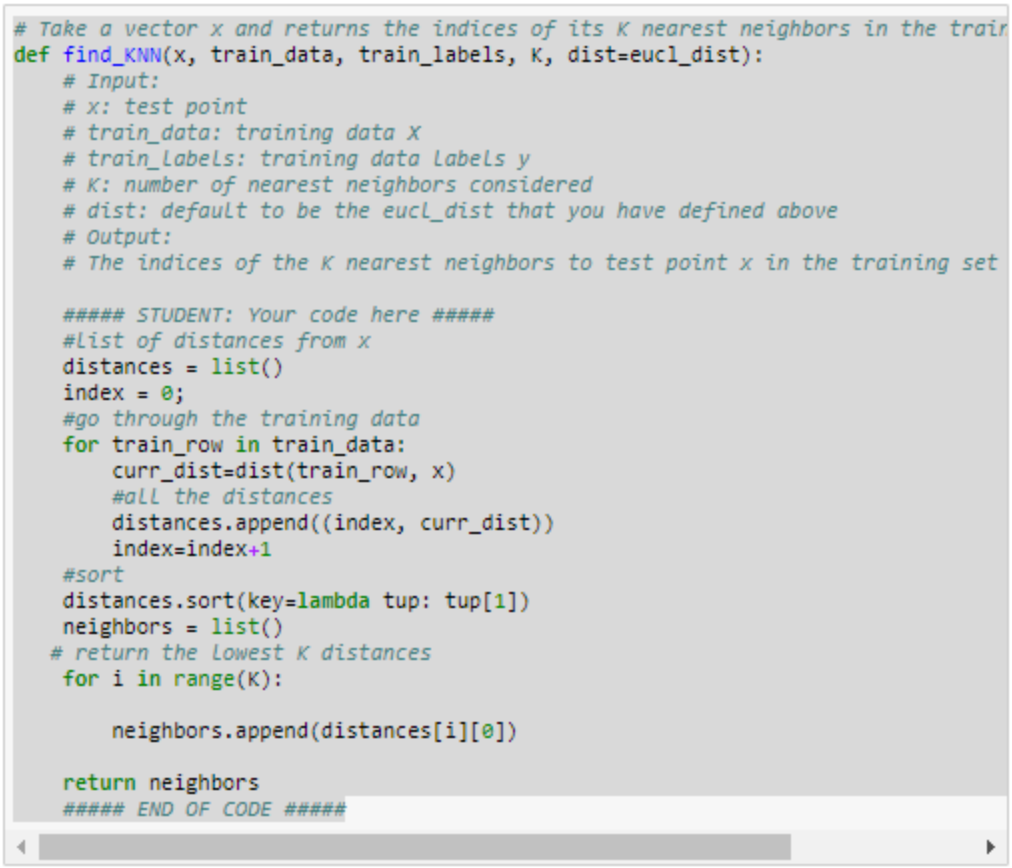
  ### STUDENT: YOUR CODE HERE

  d = len(x)
  sum = 0

  for i in range(d):
    sum = sum + pow((x[i]-y[i]),2)

  sum = math.sqrt(sum)
  return sum
CODE ENDS

b.



```
# Take a vector x and returns the indices of its K nearest neighbors in the trair
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 #####
    #list of distances from x
    distances = list()
    index = 0;
    #go through the training data
    for train_row in train_data:
        curr_dist=dist(train_row, x)
        #all the distances
        distances.append((index, curr_dist))
        index=index+1
    #sort
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
   # return the lowest K distances
    for i in range(K):

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

    return neighbors
    ##### END OF CODE #####
```

# 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=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 #####
    #list of distances from x
    distances = list()
    index = 0;
    #go through the training data
    for train_row in train_data:
        curr_dist=dist(train_row, x)
        #all the distances
        distances.append((index, curr_dist))
        index=index+1
    #sort

```
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
   # return the lowest K distances
    for i in range(K):

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

    return neighbors
    ##### END OF CODE #####
```

```
# 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 #####

    #get the lowest K neighbors
    neighbors = find_KNN(x, train_data, train_labels,K,dist)

    if K is 1:
        # return the only label
        return train_labels[neighbors[0]]
    else:
        #value will be teh amount of times an number appears by index
        value = [0]*10
        #find the max alue of an item
        max =0;
        index = 0;
        #find the average of classification between K objects returned from find_
        for item in neighbors:
            value[train_labels[item]]=value[train_labels[item]]+1
        #find the number that appeared most often and return it
        for i in range(10):
            if value[i]>max:
                max = value[i]
                index = i
        return index
    ##### END OF CODE #####
```

```
# 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 #####

#get the lowest K neighbors
neighbors = find_KNN(x, train_data, train_labels,K,dist)

if K is 1:
   # return the only label
   return train_labels[neighbors[0]]
else:
   #value will be teh amount of times an number appears by index
   value = [0]*10
   #find the max alue of an item
   max =0;
   index = 0;
   #find the average of classification between K objects returned from find_KNN
   for item in neighbors:
      value[train_labels[item]]=value[train_labels[item]]+1
   #find the number that appeared most often and return it
   for i in range(10):
      if value[i]>max:
         max = value[i]
         index = i
   return index
##### END OF CODE #####
```

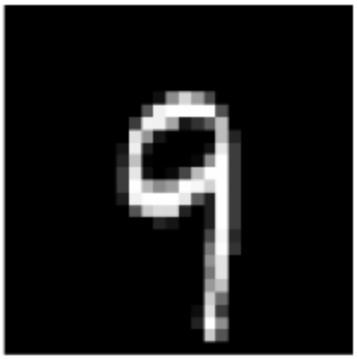
c.

```
A success case:
1-NN classification:  6
True label:  6
The test image:
5
```

```
Label 6
The corresponding nearest neighbor image:
```

```
Label 6
A failure case:
NN classification:  9
True label:  3
The test image:
```

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



d. The error for 3-NN for my algorithm was 0.076. I have a 2.56 Ghz machine. My classification time was 2548 seconds.

```
100% |########################################################################
##|

Error of nearest neighbor classifier with Euclidean distance:  0.076
Classification time (seconds) with Euclidean distance:  2548.9257085323334
```

e.

```
## 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
    d = len(x)
    sum = 0.0

    for i in range(d):
        sum = sum + abs(x[i]-y[i])

    return sum
    ### CODE ENDS
```

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
d = len(x)
sum = 0.0

for i in range(d):
sum = sum + abs(x[i]-y[i])

return sum
CODE ENDS

```
100% |########################################################################
##|

Error of nearest neighbor classifier with Manhattan distance:  0.086
Classification time (seconds) with Manhattan distance:  961.9957449436188
```

This code took 961 seconds with a 0.086 error

f.

```
## 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
    #get the lenght of x and y

    #Squared chord distance (SCD)
    #we take the square root of each point,
    #find the diffrence and square that diffrence

    sum = 0.0

    d = len(x)

    for i in range(d):
        diff = abs(math.sqrt(x[i]))-abs(math.sqrt(y[i]))
        sum = sum +pow(diff,2)

    return sum
    ### CODE ENDS
```


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
  #get the lenght of x and y

  #Squared chord distance (SCD)
  #we take the square root of each point,
  #find the diffrence and square that diffrence

  sum = 0.0

  d = len(x)

  for i in range(d):
    diff = abs(math.sqrt(x[i]))-abs(math.sqrt(y[i]))
    sum = sum +pow(diff,2)

  return sum
  ### CODE ENDS

This code had an error of 0.056, lower than Euclidian and Manhattan distance
The time was 2049 seconds

```
100% |################################################################################
##|

Error of nearest neighbor classifier with the new distance:  0.056
Classification time (seconds) with the new distance:  2049.391456604004
```

g.

```
### STUDENT: YOUR CODE HERE

# Split a dataset and solutions into k folds
dataset_split = list()
dataset_copy = deepcopy(test_data)
labels_split = list()
labels_copy = list(test_labels)
#get the size each fold should be
fold_size = int(len(test_data) / 5)

#print(dataset_copy[0])

curr_index = 0
for i in range(5):

    fold = []
    solutions = list()

    while len(solutions) < fold_size:
    #palce a random index into the return list
        index = randrange(len(labels_copy))

        fold.append(dataset_copy[index])
        dataset_copy = np.delete(dataset_copy,index,0)
        solutions.append(labels_copy.pop(index))
        curr_index +=1

    #print(len(dataset_copy))
    #print(len(solutions))
    #print(fold)
    dataset_split.append(fold)
    labels_split.append(solutions)
```

First we must split the labels and test data randomly and make sure that the indexes in the new label matrix and the new test data matix. Solutions and fold respectivley

```
    #print(len(dataset_copy))
    #print(len(solutions))
    #print(fold)
    dataset_split.append(fold)
    labels_split.append(solutions)


curr_fold = 0
k_value = 1
for fold in dataset_split:
    pbar = ProgressBar() # to show progress
    ## Predict on each test data point (and time it!)
    t_before = time.time()

    test_predictions = np.zeros(len(labels_split[0]))
    #print(dataset_split[0][0])
    for i in pbar(range(len(labels_split[0]))):
        #print(test_data[i,])
        #print(dataset_split[i,])
        #print(labels_split[i])

        test_predictions[i] = KNN_classifier(fold[i],train_data,train_labels,curr_fold,eucl_dist)

    t_after = time.time()

    ## Compute the error
    err_positions = np.not_equal(test_predictions, labels_split[curr_fold])
    error = float(np.sum(err_positions))/len(labels_split[curr_fold])

    print("Error of nearest neighbor classifier with the new distance: ", error)
    print("Classification time (seconds) with the new distance: ", t_after - t_before)
    curr_fold = curr_fold+1
    k_value = k_value +2
```

The we test different K values with one of the 5 folds that we have created. Increasing K by two each time

STUDENT: YOUR CODE HERE

```
# Split a dataset and solutions into k folds
dataset_split = list()
dataset_copy = deepcopy(test_data)
labels_split = list()
labels_copy = list(test_labels)
#get the size each fold should be
fold_size = int(len(test_data) / 5)

#print(dataset_copy[0])

curr_index = 0
for i in range(5):

   fold = []
   solutions = list()

   while len(solutions) < fold_size:
   #palce a random index into the return list
      index = randrange(len(labels_copy))
```

```
            fold.append(dataset_copy[index])
            dataset_copy = np.delete(dataset_copy,index,0)
            solutions.append(labels_copy.pop(index))
            curr_index +=1

        #print(len(dataset_copy))
        #print(len(solutions))
        #print(fold)
        dataset_split.append(fold)
        labels_split.append(solutions)


    curr_fold = 0
    k_value = 1
    for fold in dataset_split:
        pbar = ProgressBar() # to show progress
        ## Predict on each test data point (and time it!)
        t_before = time.time()

        test_predictions = np.zeros(len(labels_split[0]))
        #print(dataset_split[0][0])
        for i in pbar(range(len(labels_split[0]))):
            #print(test_data[i,])
            #print(dataset_split[i,])
            #print(labels_split[i])

            test_predictions[i] = KNN_classifier(fold[i],train_data,train_labels,curr_fold,eucl_dist)

        t_after = time.time()

        ## Compute the error
        err_positions = np.not_equal(test_predictions, labels_split[curr_fold])
        error = float(np.sum(err_positions))/len(labels_split[curr_fold])

        print("Error of nearest neighbor classifier with the new distance: ", error)
        print("Classification time (seconds) with the new distance: ", t_after - t_before)
        curr_fold = curr_fold+1
        k_value = k_value +2
```

```
100% |########################################################################|
  0% |                                                                        |

Error of nearest neighbor classifier with the new distance:  0.91
Classification time (seconds) with the new distance:  1190.7693254947662

100% |########################################################################|
  0% |                                                                        |

Error of nearest neighbor classifier with the new distance:  0.09
Classification time (seconds) with the new distance:  959.9625060558319

100% |########################################################################|
  0% |                                                                        |

Error of nearest neighbor classifier with the new distance:  0.09
Classification time (seconds) with the new distance:  461.9388930797577

100% |########################################################################|
  0% |                                                                        |

Error of nearest neighbor classifier with the new distance:  0.05
Classification time (seconds) with the new distance:  472.0826086997986

100% |########################################################################|

Error of nearest neighbor classifier with the new distance:  0.11
Classification time (seconds) with the new distance:  486.1788935661316
```

The K value increases by 2 every time. Out puts different error for each K value. The lowest error was when K was 7