Identity Authentication Using Modified k-NN algorithm

And

New York City Taxi Driving Strategy

Team 744

Table of Contents

1. Summary for Problem I

We are trying to determine people's identity using their typing patterns. We are given 3 sets of typing data: 1) 8 quotes and 4 paragraphs where the person's identity is know, 2) 6 quotes where the person's identity is unknown, 3) 3 paragraphs where the person's identity is unknown. Our goal is to match samples in categories 2 and 3 to the personal identities in category 1.

We modify the k-NN algorithm to solve this problem. We first transform the typing samples to data points by dividing typed letters into groups of two. We convert the letters into integers, and graph the ones in the category 1 on a plot. Since 11 officers participate in category 1, we get 11 clusters on the graph. We then put each data point in categories 2 and 3 as testing points into the graph, and calculate the average distance of each cluster to the testing data points. Running through all the testing data points for one person in categories 2 or 3 and adding the average distances, we get 11 numbers representing the sum of the average distances for each cluster to reach the data points in this testing sample. Due to the fact that the number of testing data points varies between testing samples, we calculate the relative frequency of the sums by dividing them over the mean of the 11 sums.

When analyzing the results, we first find the lowest relative percentage in each of the testing samples. If there is no conflict between samples, then we assign the sample to the corresponding identity with the lowest percentage. For all the conflicting samples, we compare relative frequencies and assign the testing sample with the lowest percentage to the identity.

In the end, we are able to match the samples in categories 2 and 3 to identities in category 1. We test our model with our own data, and find the model to be 66.7% accurate. Based on our model, we determine that k-NN can be further utilized in the area of pattern recognition.

2. Summary for Problem II

For this problem, we are given a big data set on taxi runs in NYC. Based on the given data, we give advice to taxi drivers for where they should drive towards in order to look for more customers from a certain location during a certain time in the day.

We first pre-organize the raw data. We determine that the distance from the taxi driver to the pickup location of the guest, the time spent on sending the guest to the destination, the fare that the guest paid for the trip, and the tip the guest paid all have an effect on the value of the customer. Thus, we assign weights to these factors and give scores to each data point using

$$Score_{time} = w_{time} \times time + w_d \times d + w_f \times fare + w_{tip} \times tip .$$

We also divide the map of NYC and nearby areas into smaller regions and give each region scores for each hour of the day by adding all the data points' scores that fall into the hour in this region.

When inputting the time and a driver's location in longitudes and latitudes, our model analyzes which region has the highest score for the driver, and advice the driver to drive to that region. We set two examples for starting location – the Statue of Liberty and the Penn Station, at 12:00am-1:00 pm – and the algorithm suggests both drivers drive to region 48359, which is located in Southern NYC.

For question 2, from the perspective of the taxi company, we suggest the drivers do the following: decrease the amount of time the taxi remains unoccupied, meet as much taxi needs as possible, and drive to Southern NYC if they don't know where to look for new customers.

3. Model for Problem I

3-a Introduction

Many methods are used for identity authentication, such as: passwords, facial recognition, and pattern recognition. Today, people's typing patterns can also be a useful tool for verification. In this question, we are given 11 people's typing practice results for 14 English quotes and 6 paragraphs composed of random characters. The results are divided into 3 categories: 1) 8 quotes and 4 paragraphs where the person's identity is known, 2) 6 quotes where the person's identity is unknown, 3) 3 paragraphs where the person's identity is unknown. We are given their WPM values, accuracies, and a screenshot of their results. Using the given identities in the first category, we are trying to match the results in the second and third categories to the identities.

3-b Assumptions and Justifications

1) Since typing patterns can verify a person's identity, we assume that when we plot the typing test results onto a graph, the data points for the same person will cluster close to each other. In addition, person 1's cluster should be relatively distinct to person 2's cluster.

3-c Symbols

No symbols are used.

3-d Building the Model

In order to use the data in the screenshots of people's typing practices, we first convert them into word documents using www.onlineocr.com. For each practice typed by each person,

we convert the letters and punctuations into integers, and store the results in CSV files because CSV is easier to program with.

We determine that grouping typed letters in groups of two and comparing the groups of different people is a sufficient way to identity people's typing patterns. For example, if Person A has a tendency to mistype "t" with "y", then this person would, likely, type "YhThe" instead of "The". In this way, Person A will have data points "Yh" and "Th" while others only have "Th". The distinct "Yh" point allows us to identify Person A.

After grouping typed letters and punctuations into groups of two, we represent them using integers relative to their positions on the keyboard. We add a string at the end of the group indicating which person typed the sample (see code in Appendix 1). For example, if Person A types "Dream big":

Dream big

↓

(30,32,A)

(32,39,A)

(39,26,A)

(26,55,A)

(55,54,A)

(54,57,A)

(57,40,A)

(40,34,A)

Figure 1: Converting Characters to Integers

In assumption 1, we assume that when we plot the typing results onto a graph, the data points for the same person will cluster close to each other. Thus, when another data point for Person A enters the graph, it will locate near the Person A's cluster. We utilize a modified k-NN (k-Nearest Neighbors) machine learning algorithm to meet our goal. Instead of counting the k nearest data points close to one data point, we calculate the distances from these points to the one data point.

We first graph the data points of the 11 officers in the given identity set. According to our assumption, that should form 11 clusters. When we put a testing data point (a group of two integers) into the graph, k-NN allows us to add distances of all the given data points within a 20 distance range to this testing data point, and we add the distances by identity. Since each person in the given identity set has a different amount of data points, we divide the sum of the distances by the number of data points in each cluster to get an average distance from a cluster to a single testing data point. Running through all the testing data points for each testing sample and adding the average distances together, we get the 11 total average distances for each sample. Because some people have more testing data points than others, thus resulting in overall larger total average distances for all the clusters, we divide each total average over the mean value of the averages to get a relative percentage. A lower relative percentage means that the testing sample is closer to the cluster.

By comparing each testing sample's relative percentages for each cluster, we are able to match the testing samples with the known identities (code in Appendix 2).

3-e Results from the Model

When analyzing the results, we first find the lowest relative percentages in each of the testing samples. If there is no conflict between samples—for example, if sample A's lowest relative percentage is Person 1 while all the other samples' are Person 2—then we assign the samples to the corresponding identities (assign sample A as Person 1). For all the conflicting samples, we compare their relative frequencies for this identity and assign the testing sample with the lowest percentage to the identity.

Matching results from the Model

(Full result in Appendix 3)

Participant	AI Deduction
A	Person 2
B	Person 9
C	Person 5
D	Person 3
E	Person 4
F	Person 11
G	Person 6
H	Person 8
I	Person 10
J	Person 7
K	Person 1
Q	Person 10
R	Person 11
S	Person 8
T	Person 1
U	Person 6
V	Person 5

W	Person 3
X	Person 7
Y	Person 9
Z	Person 4

Table 1: Results for P1

3-f Testing the Model

In order to test this model, 3 members of our team generated training sets and test sets. Each of us typed 5 same quotes on https://www.keyhero.com and another different quote for testing. The numbers in the chart are the sums of the distances from one given cluster (K, J, or Y) to the data points in the testing samples. A lower value means a closer connection between the given cluster and the testing sample.

Participants	Test 1 (K's)	Test 2 (Y's)	Test 3 (J's)	Training Size
K	1044.19	1403.56	1507.04	1845
J	1061.74	1422.05	1525.74	1925
Y	1046.89	1403.14	1506.75	1948
Testing Size	200	242	257	/
AI Deduction	K	Y	Y	/

Table 2: Testing P1's model

In this case, our model is 66.7% accurate.

The model identifies J's testing sample to Y. This may be due to the fact that J used Y's computer for typing the test sample and he is not familiar with Y's keyboard.

3-g Strengths and Weaknesses

Strength:

1) Our model analyzes people's typing patterns for every 2 letters, which reflects accuracy.
2) Our model allows computer to do the work instead of human, which eliminates human error.

3) Our modified k-NN algorithm uses weight, which is able to cancel out the imbalance of the training set size.

Weaknesses:

1) k-NN algorithm is lazy learning, and its classification is not normalized.
2) Our model does not utilize the typing speeds and accuracy percentages of the typing samples, which one normally associates with distinguishing different people's typing samples. According to MOOC's personal identification technology [2], there is a relationship between the time one hits two consecutive keys to determining one's identity. Thus, the neglect of these provided condition is a weakness of our model.

4. Model for Problem II

4-a Introduction

Taxis have become an indispensable kind of passenger transportation in the cities. However, the multi-point and multi-stage development of the cities and difference in regional development lead to the difference in distribution of the taxis and the need for them. Therefore, when taxis are vacant, where drivers should go becomes a widely discussed issue in order to improve the service level of taxis and more importantly, the efficiency of the social transportation. In the first part of this question, given the taxi trip record data for Green Taxi in New York City, we manage to figure out the optimal decision for the drivers about where to go when their cars are vacant. In the second part, as a head of a taxi company, we try to give advice to the drivers, combining the conclusion we obtain from the first question and consideration on both social business factors.

4-b Assumptions and Justifications

1) For question one, we assume that the drivers are looking for gaining the most money with the least time and distance driven. We assume that different regions have different needs for taxis. Also, we assume that the distance between the pickup location and destination is a straight line, which can be determined using longitude and latitude of the locations.
2) For problem two, we assume that the taxi company wants to fulfill the goal of letting more people use their taxis while ignoring the driver's personal benefits. We assume that they want to decrease the amount of time of taxi running without a customer.

4-c Symbols

w_{time}	Weight applied to time
w_d	Weight applied to distance
w_f	Weight applied to fare
w_{tip}	Weight applied to tip
d	Distance to pick up the guest
$lat_{\max}$	Maximum latitude in given data
$lat_{\min}$	Minimum latitude in given data
$long_{\max}$	Maximum longitude in given data
$long_{\min}$	Minimum longitude in given data
s	The length of the divided region

Table 3: Symbols used in P2

4-d Building the Model

In this problem, we are given data on NYC's taxi service. The data includes information about the pickup location, the dropoff location, the number of passenger, and fare amount. We give each pickup point a value, considering the key parts below: the distance from the taxi driver to the pickup location of the guest, the time spent on sending the guest to destination, the fare that the guest paid for the trip, and the tip the guest paid.

We consider these key parts to have different weights. We determine that the fare is the most important factor: how much the trip is worth determines how much money the drivers can make. The second important factor is the time of the trip. We give this a negative factor, since the longer the traveling time is, the less efficient the driver will be (the fare per mile gets lower when drivers drive longer distances). The distance to pickup location is the third most important factor, because this influences the efficiency of the driver by making him/her waste time. The

least important factor is the tip, because some guest would pay tips and some would not--this is mostly random.

With the above reasons, we get the following weights for time, distance, fare, and tip.

$$w_f = 10,\ w_d = -8, w_{time} = -7,\ w_{tip} = 3$$

Another part of the pre-organization of the given data is to divide the area of the whole map. Using the maximum and minimum of the longitude and latitude of the locations, we set the frame of the map, and divided the map into many little square regions with a 0.02 difference in both latitude and longitude, which is approximately equivalent to 2.22 kilometers. Thus, the map is divided with:

$$(lat_{\max} - lat_{\min}) \div s \text{ rows}$$

and $(long_{\max} - long_{\min}) \div s$ columns

After dividing the map, we give each region a score. We also take into account the time when the taxi is unoccupied. Different times during the day can have different effects on the need for taxi.

$$Score_{time} = w_{time} \times time + w_d \times d + w_f \times fare + w_{tip} \times tip \quad \text{Equation 1}$$

We then calculate the score for each potential customer in each region. Adding the scores of all guests in the region, we get a score for the whole region.

$$Score_{region,time} = \sum Score_{all\ data\ in\ the\ region} \quad \text{Equation 2}$$

After inputting the location of the driver (in latitude and longitude), the program compares the scores of the all the regions on the map, and returns the one with the highest score. That's the region we advise the driver to drive to (see code in Appendix 5).

4-e Results from the Model

Since our model depends on the location of the driver and the time, we do not have any results, other than our algorithm, to display. To demonstrate how our model works we put in 2 popular locations in NYC as the location of the driver, and test which region the driver should drive to.

Location Name	(Longitude, Latitude)	Time	Region	Score
The Statue of Liberty	(-74.0445, 40.6892)	12:00am-1:00pm	48359	87210
Penn Station	(-76.2970, 40.2131)	12:00am-1:00pm	48359	87347

Table 4: Results for P2

To look for a general trend, we set the location of the driver at the Statue of Liberty and run through all hours of the day. We find that for most time periods, region 48359 appears to have the highest score. We find out the location of region 48359 on Google Earth.

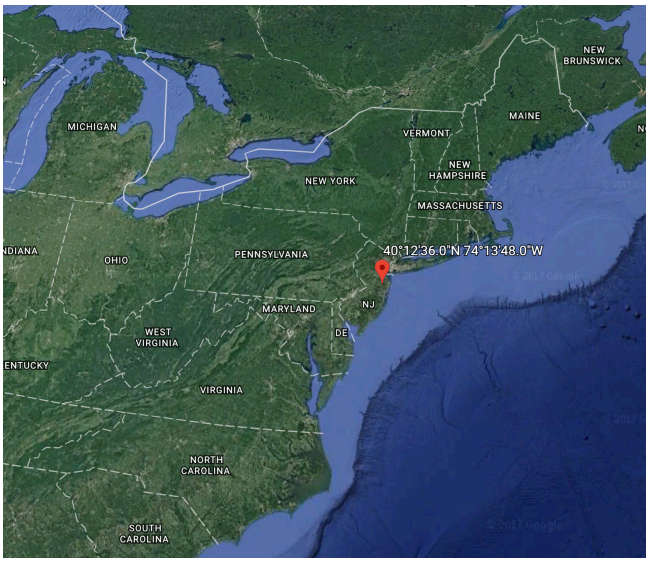

Figure 2: The location of region 48359

4-f Testing the Model

Due to time limits, we are not able to find similar data to the one provided by the problem to test our model.

4-g Strengths and Weakness

Strengths:

1) Our model is very flexible, because we used a lot of variables that can be changed, including the weight, the separation (the width) of the regions. Also, we can add time into account quickly.

Weaknesses:

1) Our weight is not based on scientific research; we decide them based on our own logic.
2) The divided regions do not represent the real world effectively enough. Our algorithm gives a region code, while the drivers in the real world would just want to go towards specific locations such as the center of the city.

4-h Solution to Question 2: A Letter to Taxi Drivers

Dear Drivers in Green Taxi,

Thank you for your dedication, and hope you are making profits everyday. As the head of Green Taxi, I want to deliver to you some of the decisions the company made recently.

Firstly, we are going to emphasize the new goal of minimizing the time your taxi remains unoccupied. Unoccupied taxies mean inefficiency to our company, and it is a disadvantage for you as well. Driving for customer means making a profit, while running with an unoccupied taxi is spending resources such as gasoline, electricity, and your time. I am sure that you all want to earn as much as possible, and the company wants to do so, too. Thus, please be consciously looking for customers when you don't have one.

Secondly, the company wants to build a reputation of being "always there" for the customers. What this means is that, when you see a person trying to find a taxi nearby, please make every effort to get to the person as fast as possible. Even if this person is fairly far from your location but no one has answered his/her request for taxi, we encourage you to drive to that location. If we can respond to people's needs for taxi faster, we can develop a good reputation and an advantage in the market.

Finally, if you just finish a drive and is looking for new customer, we advise you to drive towards South Manhattan. With a math model, we determined that that you will likely encounter a good deal along the way.

Best,

Green Taxi

5. Citations

[1] Andrew Maas, Chris Heather, Chuong Do, Relly Brandman, Daphne Koller, and Andrew Ng, *MOOCs and Technology to Advance Learning and Learning Research*, http://ai.stanford.edu/~amaas/papers/amaas_mooc_verified.pdf

[2] oYabea, cnblogs, *Machine Learning—K Nearest Neighbor (KNN) Algorithm*, http://www.cnblogs.com/ybjourney/p/4702562.html

[3] C H Chen, *Handbook of Patter Recognition and Computer Vision*, 5th Edition, 2015, https://books.google.com/books?hl=en&lr=&id=IWzFCwAAQBAJ&oi=fnd&pg=PR7&dq=pattern+recognition&ots=dy5gsy0nSr&sig=jzxIG_a0d6Z4zaTojE4c08inLjY#v=onepage&q&f=false

[4] Jay Belanger, *Write Right for the American Mathematical Contest in Modeling*, Higher Education Press, 2013

[5] Frank R. Giordano, William P. Fox, Steven B. Horton, *A First Course in Mathematical Modeling*, China Machine Press, 2015

[6] Ye Xianyan, *The Model and Facter Analysis of Effect on the Urban Taxi Ridership Based on Geographically Weighted Regression*, Southwest Jiaotong University Master Degree Thesis, May 2017

[7] Brunsdon, C., Fotheringham, A. S., & Charlton, M. E. *Geographically weighted regression: a method for exploring spatial nonstantionarity*. Geograph- ical Analysis, 1996,28(4), 281-298

Appendix 1: code for converting letters to integers

```
import csv
import xlwt
import os
import sys
location='Person01.txt'
path='/Users/***/Desktop/AoCMMTest/Given'
def readstringfiles(name):
    arr=[]
    fp = open(path+'/'+name, 'r')
    n=str(name[-5])
    array=[]
    for lines in fp.readlines():
        string=lines
        for x in range(0,len(string)-2):
            array.append(string[x])
            array.append(string[x+1])
            array.append(string[x+2])
            array.append(n)
            arr.append(array)
            array=[]
    fp.close()
    return arr
def letter2number(s):
    num=[]
    letter="1234567890-=+_)(*&^%$#@!qaswedfrtghyujkiol;p[]'/.,<>? mnbvcxz"
    if s.lower() in letter:
        return letter.index(s.lower())+1
    else:
        return 0
def array2num(arr):
    for x in range(0,len(arr)):
        for y in range(0,3):
            arr[x][y]=letter2number(arr[x][y])
    return arr
def write(arr):
    '''
    writing book
    '''
    book=xlwt.Workbook(style_compression=0)
    sheet=book.add_sheet('output',cell_overwrite_ok=True)
    for x in range(0,len(arr)):
        for y in range(0,4):
            sheet.write(x,y,arr[x][y])
    book.save(path+"/"+arr[0][3]+'.csv')
def ConvertAll():
    filelist=os.listdir(path)
    for files in filelist:
        filename=os.fsdecode(files)
        if filename.endswith(".txt"):
            write(array2num(readstringfiles(files)))

#ConvertAll()
write(array2num(readstringfiles(location)))
```

Appendix 2: Python code for P1

```
import csv
import random
import math
import operator
import itertools
import os

path='/Users/****/Desktop/AoCMMTest/Given'
sample='/Users/****/Desktop/AoCMMTest/Test/quotes/K.csv'

def most_common(L):
  SL = sorted((x, i) for i, x in enumerate(L))
  groups = itertools.groupby(SL, key=operator.itemgetter(0))
  def _auxfun(g):
    item, iterable = g
    count = 0
    min_index = len(L)
    for _, where in iterable:
      count += 1
      min_index = min(min_index, where)
    return count, -min_index
  return max(groups, key=_auxfun)[0]

def loadDataset(filename, split, trainingSet , testSet):
        with open(filename, 'r') as csvfile:
            lines = csv.reader(csvfile)
            dataset = list(lines)
            for x in range(len(dataset)):
                for y in range(4):
                    if y<3:
                        dataset[x][y] = float(dataset[x][y])
                    else:
                        dataset[x][y] = (dataset[x][y])
                if random.random() < split:
                    trainingSet.append(dataset[x])
                else:
                    testSet.append(dataset[x])

def euclideanDistance(instance1, instance2, length):
        distance = 0
        for x in range(length-1):
                distance += pow((instance1[x] - instance2[x]), 2)
        return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, predictions, name,k):
        distance=[]
        length = len(testInstance)-1
        for x in range(len(trainingSet)):
          dist = euclideanDistance(testInstance, trainingSet[x], length)
          if dist <=k:
            predictions[name.index(trainingSet[x][3])]+=(dist)

def frequency(arr,string):
  count=0
```

```
  for x in range(0,len(arr)):
    if arr[x]==string:
      count+=1
  return count/len(arr)

def printResults(name,predictions,number):
  total=sum(predictions)
  for x in range(len(name)):
    print(name[x]+": "+str((predictions[x]/number[x])/(total/sum(number))))

def determine(location):
        result=[]
        trainingSet=[]
        testSet=[]
        others=[]
        split = 1
        trainingSet=[]
        testSet=[]
        name=[]
        predictions=[]
        number=[]
        filelist=os.listdir(path)
        for files in filelist:
          filename=os.fsdecode(files)
          if filename.endswith(".csv") and filename!=location:
            loadDataset(path+'/'+files, split, trainingSet, testSet)
        loadDataset(location,split,testSet,others)
        for x in range(0, len(trainingSet)):
          if trainingSet[x][3] in name:
            number[name.index(trainingSet[x][3])]+=1
          else:
            number.append(1)
            name.append(trainingSet[x][3])
            predictions.append(0)
        print('Train set: ' + repr(len(trainingSet)))
        print('Test set: ' + repr(len(testSet)))
        for x in range(len(testSet)):
              getNeighbors(trainingSet,testSet[x],predictions,name,20)
        printResults(name,predictions,number)

print(sample[-6]+sample[-5])
determine(sample)
```

Appendix 3: full result for Problem 1

Participant	Testing Size	AI Deduction	Relative percentage
A	1355	Person 2	1.0043
B	1370	Person 9	1.0019
C	1346	Person 5	1.0015
D	1441	Person 3	1.0022
E	1335	Person 4	0.9768
F	1347	Person 11	1.0047
G	1346	Person 6	0.9959
H	1343	Person 8	1.0018
I	1339	Person 10	1.0027
J	1427	Person 7	1.0022
K	1343	Person 1	1.0012
Q	322	Person 10	1.0021
R	316	Person 11	1.0011
S	314	Person 8	1.0022
T	330	Person 1	0.9996
U	312	Person 6	0.9964
V	305	Person 5	1.0007
W	311	Person 3	1.0012
X	318	Person 7	1.0012
Y	317	Person 9	1.0009
Z	320	Person 4	0.9812

Appendix 4: full results for P2

Time	Pickup Region)	Value	Latitude	Longitude
0-1	48359	111949.85479396600000	40.21312128264600	-74.2303450021617
1-2	48359	112211.01109667300000	40.21312128264600	-74.2303450021617
2-3	48359	73248.45828872830000	40.21312128264600	-74.2303450021617
3-4	48362	56361.64437394580000	40.21328821754970	-74.2300951105553
4-5	48362	57249.72493114790000	40.21328821754970	-74.2300951105553
5-6	48357	17331.24388891330000	40.21300999271020	-74.2305115965660
6-7	48359	20906.27643650770000	40.21312128264600	-74.2303450021617
7-8	48359	45420.92235307860000	40.21312128264600	-74.2303450021617
8-9	48359	79301.77904409620000	40.21312128264600	-74.2303450021617
9-10	48359	96550.20114810130000	40.21312128264600	-74.2303450021617
10-11	48359	80932.88638860390000	40.21312128264600	-74.2303450021617
11-12	48359	88915.10742165890000	40.21312128264600	-74.2303450021617
12-13	48359	87210.05336007580000	40.21312128264600	-74.2303450021617
13-14	48359	77560.33985656570000	40.21312128264600	-74.2303450021617
14-15	48359	84365.87335377320000	40.21312128264600	-74.2303450021617
15-16	48359	110128.33070740800000	40.21312128264600	-74.2303450021617
16-17	48359	89953.33234278660000	40.21312128264600	-74.2303450021617
17-18	48359	122150.51200546400000	40.21312128264600	-74.2303450021617
18-19	48359	142601.60501567000000	40.21312128264600	-74.2303450021617
19-20	48359	158730.57781477300000	40.21312128264600	-74.2303450021617
20-21	48359	167280.96579869100000	40.21312128264600	-74.2303450021617
21-22	48359	168402.46171696200000	40.21312128264600	-74.2303450021617
22-23	48359	163595.88267136600000	40.21312128264600	-74.2303450021617
23-24	48359	102264.13065182100000	40.21312128264600	-74.2303450021617

Appendix 5: code for P2

```
import csv

datalocation='/Users/******/Desktop/2016_Green_Taxi_Trip_Data.csv'
##Variables
inilong=-74.0445
inilat=40.6892
wt=-7
wd=-8
wf=10
wtip=3
sep=0.02

def change2time(string):
    t=string.split(" ")[1]
    time=int(t.split(":")[0])*60+int(t.split(":")[1])
    return time

def loadDataset(filename):
    data=[]
    with open(filename,'r') as csvfile:
        lines=csv.reader(csvfile)
        dataset=list(lines)
        for x in range(1,len(dataset)):
            data.append(dataset[x])
    return data

def value(x):
    return wd*distance[x]+fare[x]*wf+wtip*tip[x]+wt*time[x]

def cell(maxlat,minlat,maxlong,minlong,sep,time1,time2,time):
    region=[]
    for x in range(0,int((maxlong-minlong)/sep)+1):
        for y in range(0,int((maxlat-minlat)/sep)+1):
            region.append(0)
    for i in range(0,len(pulong)):
        if data[i][0]<=time1 and data[i][1]>=time2:
            region[int((int(pulong[i]-minlong)/sep)*(int((maxlat-minlat)/sep)+1)+int((pulat[i]-
minlat)/sep)+1)]+=value(i)
    return region

data=loadDataset(datalocation)
time=[]
distance=[]
fare=[]
pulong=[]
pulat=[]
tip=[]

for x in range(len(data)):
```

```
    for y in range(0,2):
        data[x][y]=change2time(str(data[x][y]))
    time.append(int(data[x][1])-int(data[x][0]))
    distance.append(((float(data[x][3])-inilat)**2+(float(data[x][2])-inilong)**2)**(1/2))
    fare.append(float(data[x][7]))
    tip.append(float(data[x][10]))
    pulong.append(float(data[x][2]))
    pulat.append(float(data[x][3]))


maxlat=max(pulat)
minlat=min(pulat)
maxlong=max(pulong)
minlong=min(pulong)
print(maxlong,minlong,maxlat,minlat)
for x in range(0,1440,60):
    region=cell(maxlat,minlat,maxlong,minlong,sep,x+60,x,data)
    print("region "+str(region.index(max(region)))+" value:"+str(max(region)))
    print("latitute"+str(minlat+sep*region.index(max(region))/((maxlong-minlong)/sep))+"
longtitute"+str(minlong+sep*(region.index(max(region))/(maxlat/sep-minlat/sep))))
```