Location Identification Problem

Author: Wei Liu

Abstract

In this short paper, we analyzed the possibility of using the past record of user's GPS location to predict his/her preference toward different establishments. The given data is the GPS locations without timestamp and also the locations and types of establishments we are interested in. We first pre-processed and explored the data and abstract useful information from the users past GPS record with different algorithms. At last, we discussed the conclusion and future work needed to prompt our work, and strategies to enhance the performance of our model.

1. Introduction

Wearable devices have wide applications from health monitoring system to intelligent agents. Wearable devices are able to collect the users GPS information, and then use those data to interactively assist the user in a variety of tasks in everyday life. Through the locations information, the intelligent agents could potentially set up a predictive model, which is able to predict the future physical location or recommend new locations to the user.

To make such a wearable device be smart, first thing is to acknowledge how it could help the users to make better decisions. The following are the scenario we could imagine when an intelligent agents, which has learnt the user's frequented location preference, could interactively help the users in daily life: 1) the device could automatically display a shopping list when a user approaches a grocery shop; 2) if the user loves Chinese cuisine, the device could recommend new restaurant nearby based on the user's past visited history; 3) the device could suggest the users go to a jogging the next day, since the device has observed the user had routine running activity on Sunday, and also the device knows the next day will be lovely weather; 4) the device could predict you are going to work at 7:30am in the morning, then it could remind you that there was a traffic jam on the highway, thus you could choose alternative route; 5) the device is able to speculate the user's job or identity, e.g., if the user's location mainly distributed around an university campus, he/she has high possibility to be a college student.

Learning from the past locations is the essential part to build such an intelligent system. Most wearable devices has GPS installed and thus could gather the user's locations information at a specific frequency. In such a system, we could collection the user's physical location in a time sequence. Time is also an essential part in the <location, time> pairs, since

we can calculate many other information from time, for instance, the duration time the user stays at a spot, the longer time users spend may indicates he/she has more interest in that spot. We can also obtain the user's speed at each time, if we have other information (e.g., heart beat rate), we will be able to infer that whether the user is driving, running, walking, sitting or taking indoor exercises.

To build such an intelligent system, the recognition of the common people's routing life is also important since it could help our analysis of the GPS data. For most people have daily routine life, we could expect that the GPS locations data would not be highly skewed distributed in both space and time domain. Most of people shuttle between work place and home in work days, and then in weekend, the user might has higher possibility to appear in grocery stores, parks or other entertainment places. So, we could expect several clusters on the map, while each cluster represents work place, home or the park/grocery store the user visited. Beyond that, we could also expect the distance between each clusters would be much larger than the radius of the clusters. It was reported that Americans spent 100 hours per year for commute on average. Assume the average driving speed is 50 km/h, and there are ~260 work days each year, we could infer that the average distance from home to work is about 19.2 km.

If we use bright dots to represent GPS location to reveal a common person's daily activity, and the brightness represents his/her staying time, then for most people, the physical appearing locations on the map would be 2~4 bright dots sparsely distributed, and several other dimmer dots represents the grocery stores, restaurants, gas station and so on. While for college students, it might be a bigger dot represents the dorm, classrooms and dining hall, which are more closely located. For a postman, the dots would be sparser and evenly distributed in a large range. From those feathers of GPS location distribution, we could infer the user's career identity.

2. Data pre-processing and exploring

With all the above premise and inference in mind, we could start to analyze the GPS data we were given. There are 160 points of GPS data are given, which represents the user Jone's location information in the past 48 hours, but we don't have the time stamp for those data.

The format of GPS is very clean and easy to process. The information of each establishments were also given, include the name, type and the GPS location of the establishments. However, those data are not as clean as user_locations, thus we need to abstract useful information from the mass. For the extensible usage in the future, I build an establishment class, which holds all the information an establishment should have, and we could add more feathers if necessary.

	establishment_latitude	establishment_longitude	establishment_name	establishment_types
0	35.7795897	-78.6381787	Raleigh	locality
1	35.7992765	-78.6899516	Carlyle Campbell Library	library
2	35.79958279999999	-78.6908077	Frankie G. Weems Art Gallery	art_gallery
3	35.7989103	-78.6909706	Martin Lot Staff	premise
4	35.823483	-78.8255621	Morrisville	locality
5	35.8577575	-78.8359571	UNC Health Care's Morrisville campus	point_of_interest
6	35.8577575	-78.8359571	ISD UNC Health Care	point_of_interest
7	35.79154	-78.7811169	Cary	locality
8	35.7896757	-78.8700624	Starbucks	cafe
9	35.7892183	-78.870043	Biscuitville	cafe
10	35.79154	-78.7811169	Cary	locality

Fig.1 The pre-processed establishments dataframe

Fig.1 shows the dataframe after treatment. From the dataframe, we observed some of the points have overlapped. This could be treated in the following three cases:

1) Data error, one of the duplicated locations has a false GPS location or false name, this case needs to be corrected.

2) Those two establishments are in the same building, and this is very common in reality. In this case, we could only use the establishment's GPS information rather than names, and types, because we have no way to come up which establishment the user have been, unless we could infer the establishment from other clues or the user tells us directly. This case would be common in big city regions, e.g., Manhattan of NYC, where many companies and facilities share the same building.

3) Some of the establishment could be included into its higher level name, for example, a GPS of a university could covers its GPS of library, or a GPS of a branch of a company would be the same with its upper level company name. In this case, we can zoom in or zoom out the structure of the establishment and pick the name or the type that most interest us.

```
list_estabs = reader.get_establishments()
print(str(list_estabs[0]))
```

```
duplicated cords found: Estab_name: ISD UNC Health Care; types: point_of_interest; GPS: (35.8577575, -78.8359571)
duplicated cords found: Estab_name: Cary; types: locality; GPS: (35.79154, -78.7811169)
duplicated cords found: Estab_name: Morrisville; types: locality; GPS: (35.823483, -78.8255621)
Estab_name: Raleigh; types: locality; GPS: (35.7795897, -78.6381787)
```

Fig. 2. Duplicated GPS of establishment detected and displayed

Fig.2 shows the output of those duplicated GPS locations in the establishements collections. Further treatment maybe needed depend different cases we have analyzed. In this paper, those duplicated data was deleted and only keep one copy.

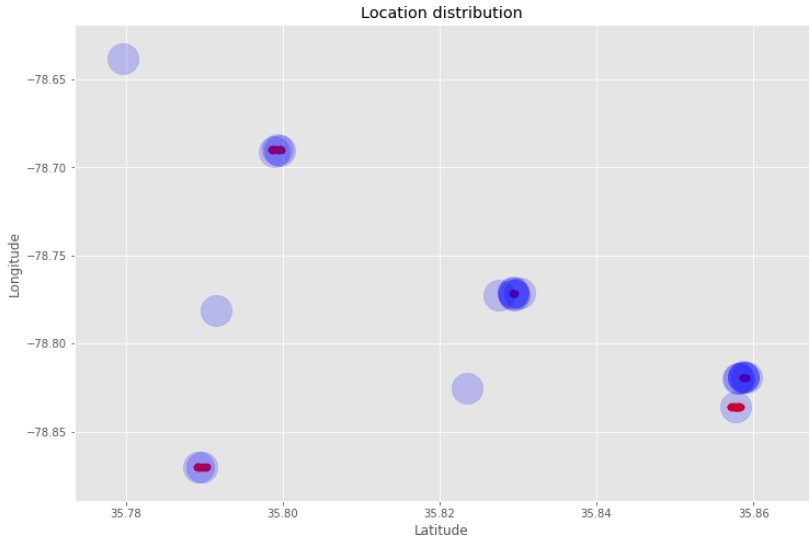


Fig.3 Data clusters on 2D map

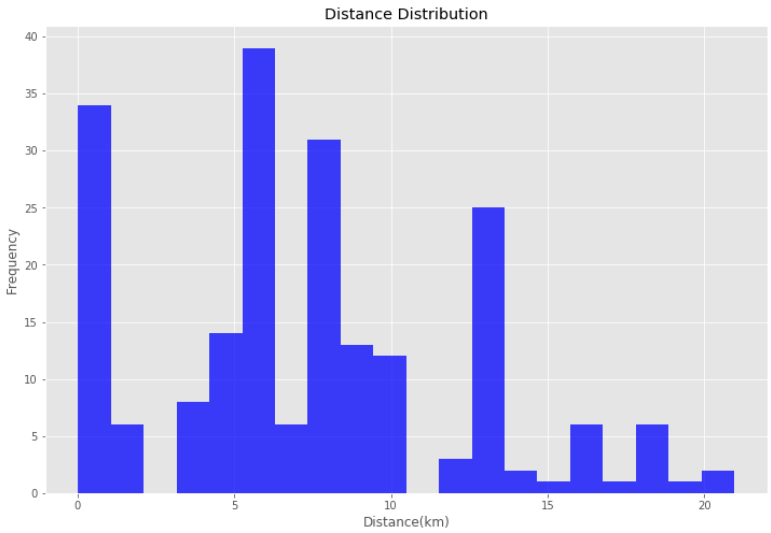


Fig. 4 Histogram of the distances between establishmens in dataset

Fig.3 is the plot of all the data points on the map. The bigger blue translucent dots represent the establishments we are interested in, while the red dots represent the GPS trace of the user. We could see that our preliminary assumption on the GPS point's distribution is validated. The data was sparsely divided into clusters, each cluster have a large distance with other clusters compares with the radius of the cluster.

Fig.4 is the distance distribution of the different establishments we are interested in. We can see the data is skewed distributed to the left of the center. We could imagine that the data would be more skewed to the left in big cities, and skewed to the right in rural area. If the distances between establishments are mostly distributed to the left, then this may bring extra difficulties to our clustering algorithms, e.g., k-means.

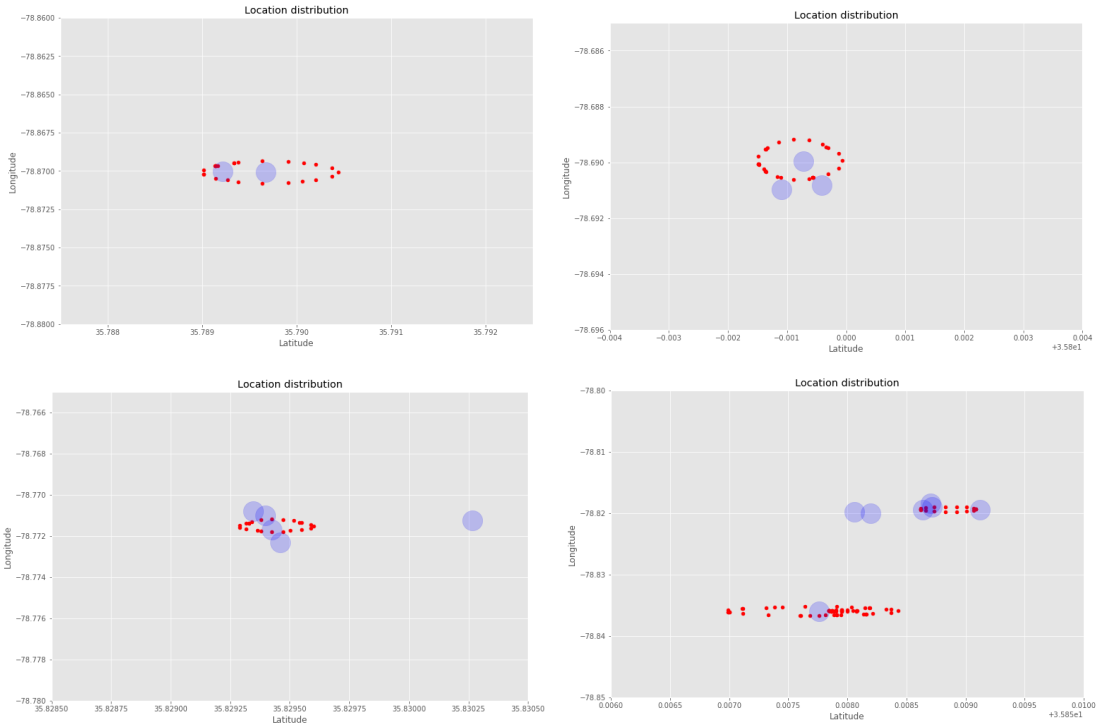


Fig. 4 Zoomed in the data cluster

Fig.4 is the zoomed in distributions of the GPS record of our user John. It is strange that the GPS shows that John seems have been walking in a circular way. If the GPS data comes from wearable devices, e.g., a watch or phone, the data may have a higher chance to be in Gaussian distribution considering the GPS drift noise. From the fig.4 we also observed that the clusters are clearly separable, even in this fine granularity level.

3. Model/Algorithm/Method

After the preliminary understanding of the data, we now start exploring of the algorithms we can use. The model should be able to predict the next stop of the user John, solely based on those 160 data points in the past 48 hours. Without the timestamp for each of GPS data, we will lose a lot of valuable information, but some of rules can still lead our analysis. The first thing we want to know is that establishments John has been to, the higher visited times and

longer stay time in the establishment, and the higher possibility of John will re-visit it again. For example, it is most likely John would visit his top favorite store rather than other stores he seldom entered. Also, the distance of John to each of those establishments is another factor for consideration. For example, if John is at a location very near a grocery store he seldom visited, than it is highly likely John is going to visit this store rather than his favorite store. Thirdly, if John has the same favorable level toward a café and a library, and now John stands in the middle between those two spots, the algorithms could give the same score to the café and the library. If we were given the current time, e.g., it is lunch time, and then the chance of entering the café would be much higher. The type of the establishment can provide little information for our decision making process without knowing the current time.

To find the most significant establishments for John, we built a function based on the distances of each of the establishment and all the GPS points. In this method, we assign each of the GPS point to its closest establishment. This is the same idea with the k-means method, where we only do the first step of k-means without the centroid update. Table 1 depicted this algorithms in pseudo code.

Table 1 Algorithm 1 to find the most important establishment for the user

Algorithm 1
Initialize cluster using the establishment's location *For each GPS record:* *Compute the distances of this point to all the centroids* *Assign this point to the centroid with the minimum distance* *Rank the centroids with the number of those points*

Table 2 Establishments and visited time

Name of the establishments	Cluster size
UNC Health Care's Morrisville campus	63
Extended Stay America Raleigh - Cary - Harrison Ave.	17
Carlyle Campbell Library	16
Biscuitville	14
Starbucks	11
Waffle House	8
Frankie G. Weems Art Gallery	7
Airport Blvd at Aerial Center Pkwy (Waffle House)	7
Martin Lot Staff	7
CapriFlavors	5
Hampton Inn Raleigh-Durham Airport	5

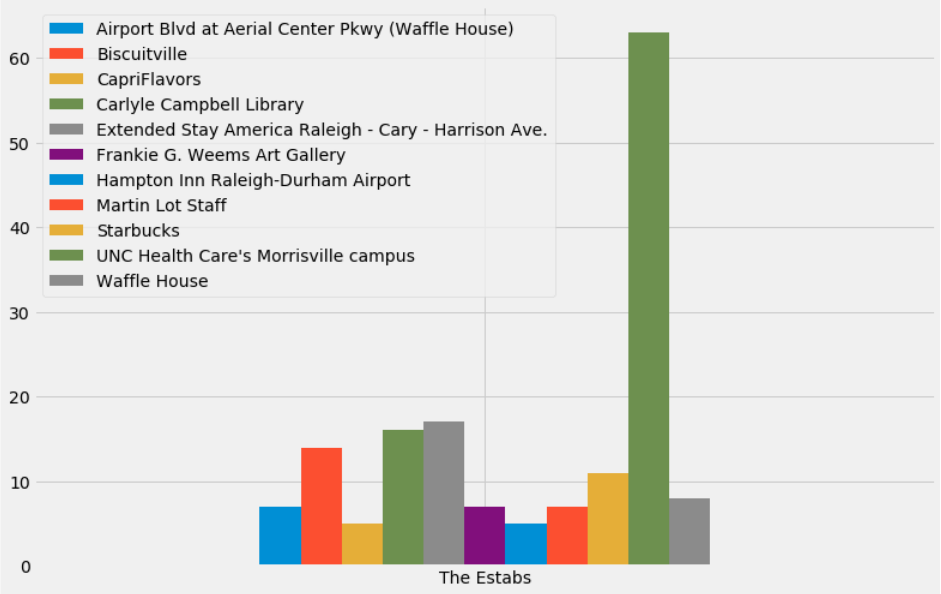


Fig.5 Histogram of the establishments and their cluster size

The most intuitive way of interpreting the cluster size is the importance of that cluster. Because we don't know the timestamp, we assume those data points were sampled evenly at same time interval in the last 48 hours. So, the importance of an establishment would be positively correlated to its cluster size. Table 4 is the output of the algorithm 1. Fig.5 is the histogram of those establishments with their cluster size.

Table 3 Algorithm 2 to find the most important establishment for the user

Algorithm 2
Initialize cluster using the establishment's location
Repeat
For each GPS record:
Compute the distances of this point to all the centroids
Assign this point to the centroid with the minimum distance
Recalculate the centroid based the new cluster
Until the centroid not change
Assign all the points in one cluster to its nearest neighbor of establishment

Table 2 is the pseudo code of applying complete k-means algorithm on the same datset. As we know, k-means working not well if the data distribution has the following features:

1) The clusters are not sphere shape;
2) The points are distributed in the way that cluster has large variety of density;
3) The initial seed points are poorly chosen;

For our algorithm, we choose the establishments centers as the seed, thus we could greatly improve the robust of the program. This will be essential, since for this specific

problem, the GPS points would be magnitude larger than the establishment points. We will have a higher chance to select the initial seed. In this program, I added the function to double check each radius of the clusters, thus to make sure we had the global optimized solution. Also, we use the physical distance of two GPS points as our distance function rather than the Euclidian distance of latitude and longitude pairs in algorithm 2.

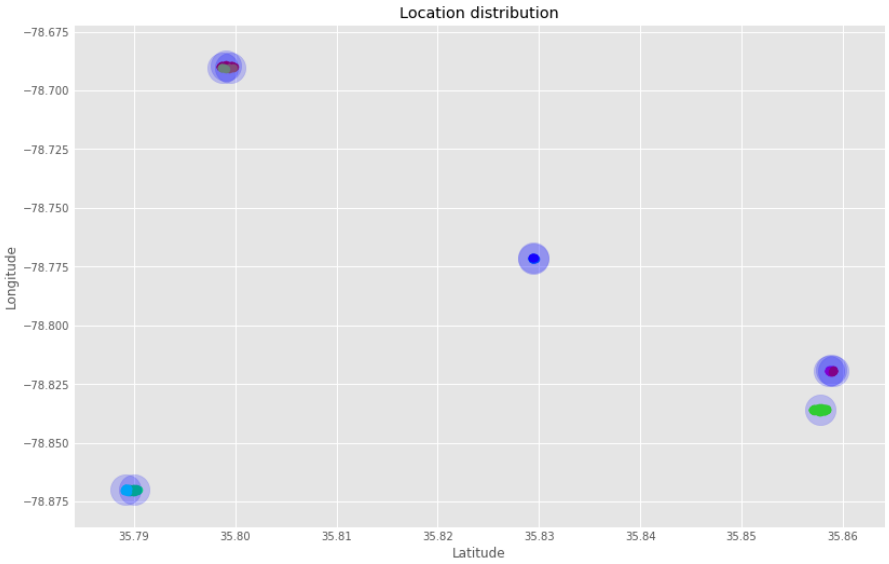


Fig. 6 Clusters after running k-means algorithm

Fig.6 is the plot after running the algorithms 2, we noticed that each cluster was marked with different colors. In fig.6, we also deleted the establishments that cluster size equals zero.

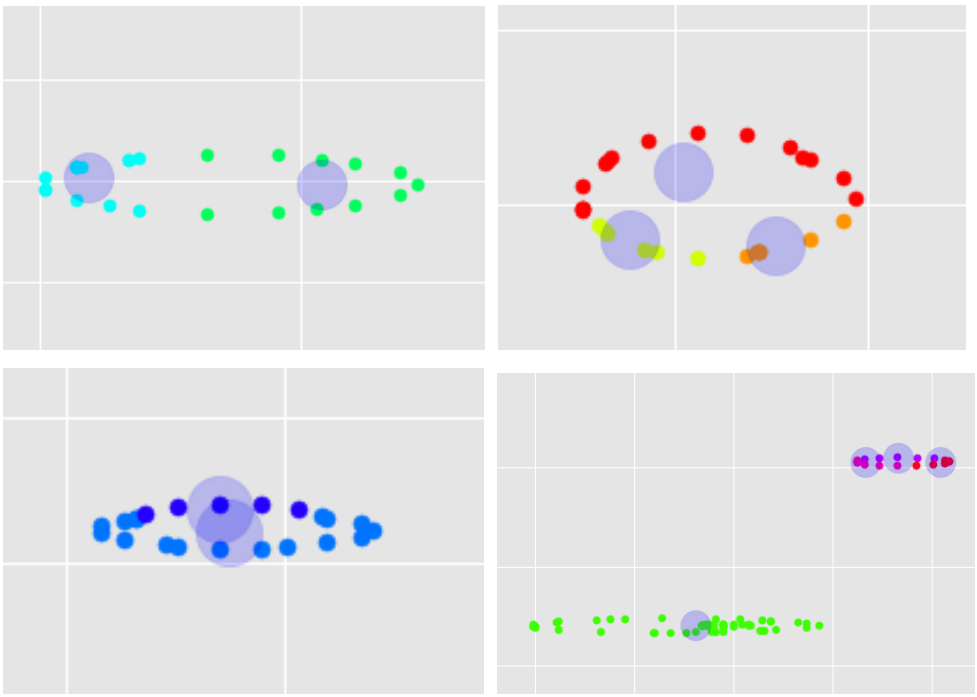


Fig. 7 Zoom in graph for each cluster

Fig.7 is the zoomed in graph for each cluster. We could see the clear boundary for each cluster around each establishment.

Table 4 Cluster radius from algorithm 2

Radius of each cluster (km)
0.10335779415809133
0.04632939844635364
0.04264085028208585
0.09233317704947942
0.08236399876431565
0.074870370812517
0.022857645183081133
0.011635469032824917
0.020711604244885863
0.031151736689348022
0.02926714038702704

Table 4 is the radius (in km) for each of those clusters; we can confirm that the algorithm has converged to the global minimum. The maximum radius found is ~100m, which makes sense in a person's daily activities.

Since we have added the centroid-update phase in algorithm 2, so, the centroid will have a shift toward the original establishment location. Through Table 5, we can see this shift is negligible.

Table 5 Comparison of the updated centroid with the original establishment centers

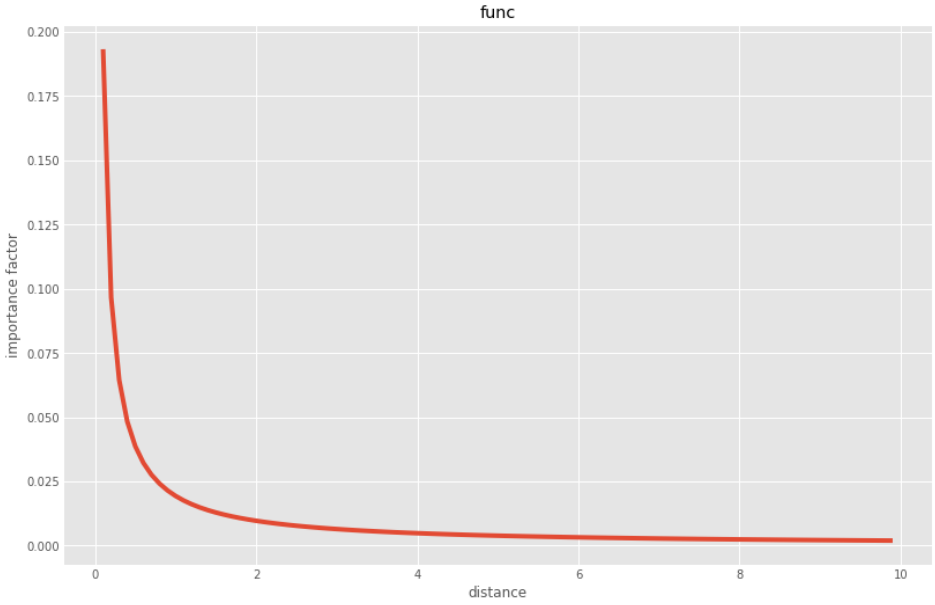
Updated centroids	Establishment centers
[35.79904067, -78.68963169],	[35.7795897, -78.6381787],
[35.79952484, -78.69048433],	[35.7992765, -78.6899516],
[35.7987671 , -78.69041078],	[35.7995827, -78.6908077],
[35.85780888, -78.83597282],	[35.823483, -78.8255621],
[35.79007535, -78.87007741],	[35.79154, -78.7811169],
[35.78918686, -78.86992024],	[35.7896757, -78.8700624],
[35.82943708, -78.77157353],	[35.7892183, -78.870043],
[35.82942633, -78.77123347],	[35.8302672, -78.771261],
[35.8588325 , -78.81899968],	[35.827521, -78.7724144],
[35.85866709, -78.81946191],	[35.8581983, -78.8199726],
[35.85904663, -78.81944097],	[35.8587064, -78.8184502],

The output of the algorithm 2 is exactly the same with the algorithm 1. This is not surprising, because all the clusters are dense and we have initialized the seed with the same

value. But for larger dataset, the output result may different, since the centroids would shift more drastically.

After clustering, we can estimate the importance of each establishment to John. For this part, we used a function:

Now we consider the current distance of John to each of the establishments.



4. Discussion and future work

have the potential to act as intelligent agents in everyday life and assist the user in a variety of tasks, using context to determine how to act. Location is the most common form of context used by these agents to determine the user's task.

However, another potential use of location context is the creation of a predictive model of the user's future movements.

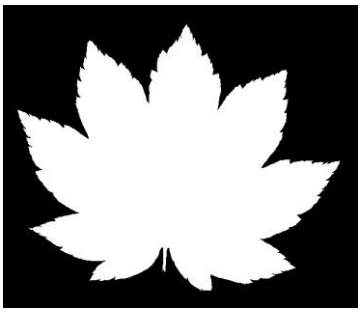
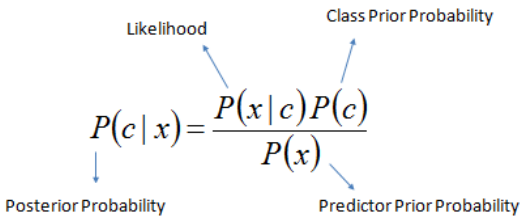


Figure 1: A leaf shape image for classification

The training data and test data I will use can be obtained from Kaggle. There are in total 192 attributes for each instance. In the training data, there are 1584 instances. In this project, different classifiers for this plants species classification/identification problem are built by plain python code, python sklearn package and also WEKA respectively. Also, the classifiers will be applied on the test dataset, and the performance of different classifiers will be evaluated. We will use the Holdout to estimate the overall accuracy for each classifier. The training set is splited by a fixed ratio. We use the larger portion as the training dataset and the smaller dataset for testing. Through this project, my goal is 1) deepen understanding to the algorithms naive Bayes and decision tree; 2) practice using the data mining tool WEKA; and 3) practice my coding skills with python.

5. Model/Algorithm/Method

The hardest part is to build the naive Bayes and decision tree using plain code with python. Following part is a brief recap of those two algorithms.



$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Figure 2: Naive Bayes formula and interpretations

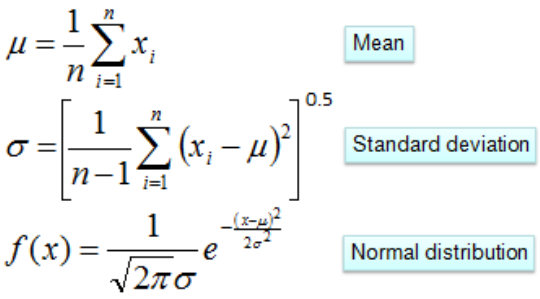


Figure 3: Probability density function for the normal distribution

Figure 2 is the formula of naive Bayes classifiers. Since in this project, all the data for attributes are continuous numbers, so we calculate the possibility assuming the numbers follows normal distribution, as shown is figure 3.

For the decision tree algorithm, we assume that each internal node has two branches. Since all the instance data for one attribute are the numerical type, so we split the instances at internal node by simple >= or < comparison. The entropy was used as the measure criteria of heterogeneity, which can be calculated as in figure 4. For each node, we choose the attribute which can generate largest information gain by comparing the entropy before and after the split.

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

Figure 4: Calculation of the entropy at nodes

For the classifiers test with WEKA, we have to 1) converting .csv file to .arff file, and 2) modifying the data type of column attributes (from string to nominal), otherwise the "incompatible data type" problem will happen for further modeling.

6. Results and Conclusions

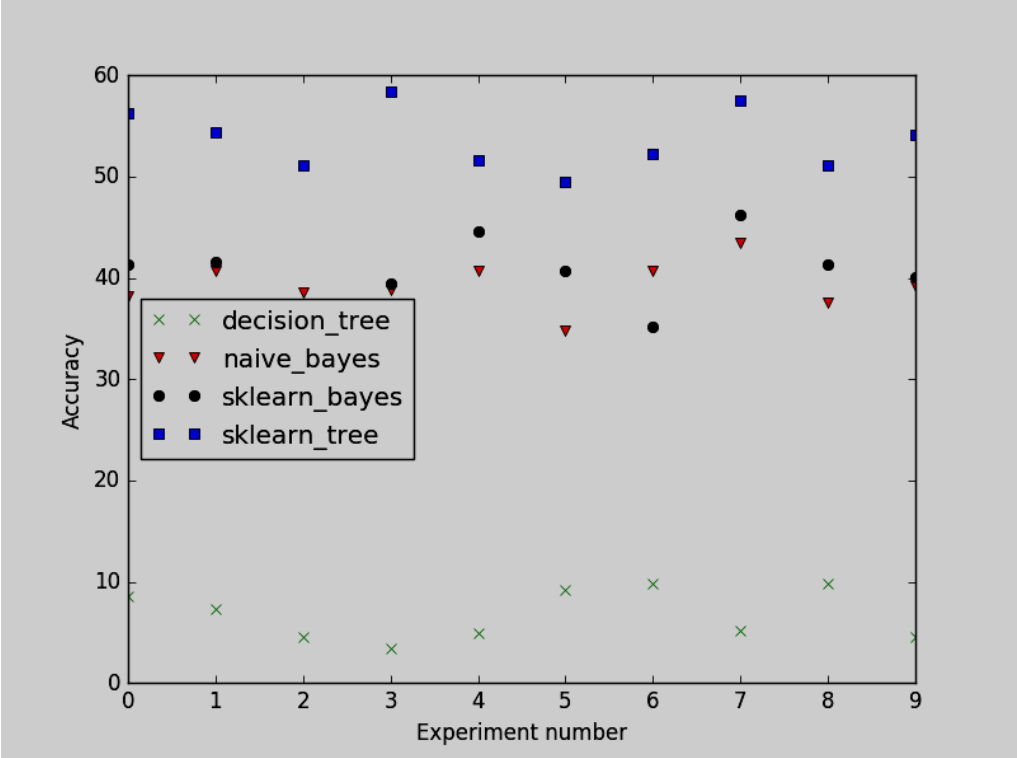


Figure 5: Comparison of accuracy of different algorithms/methods

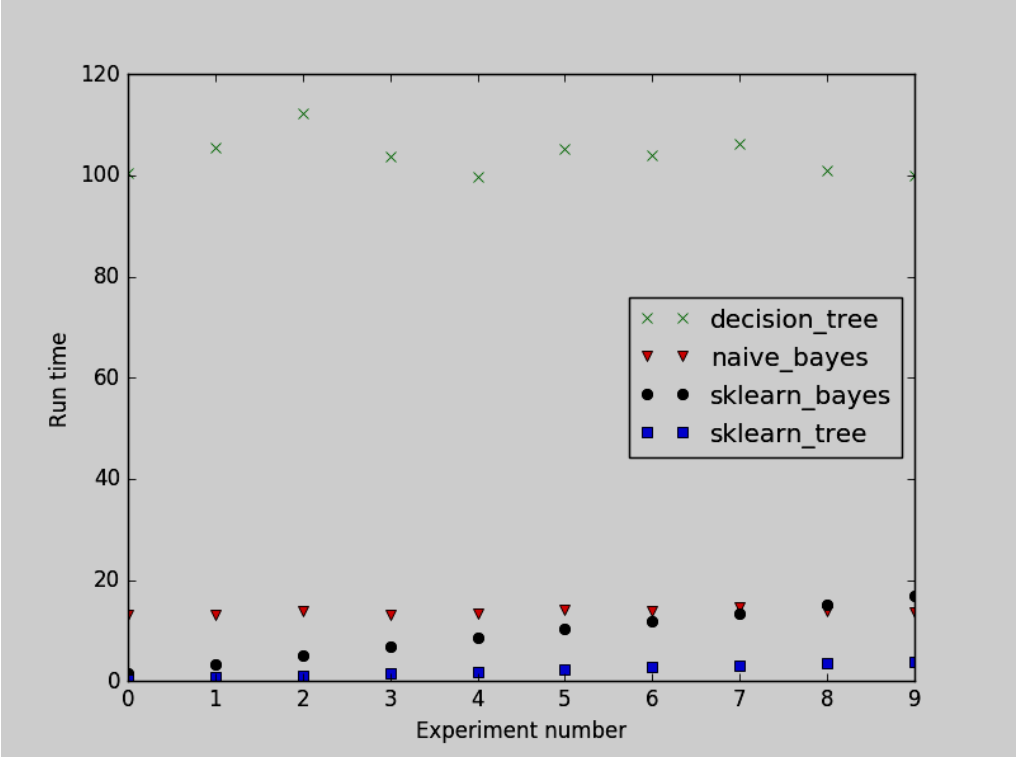


Figure 6: Comparison of runtime of different algorithms/methods

The implemented of naive Bayes and decision tree by sklearn tool seems very concise and simple. After the training dataset and testing dataset get prepared, you can call those methods like the following:

```
from sklearn.naive_bayes import GaussianNB
Y = np.array([instance[0] for instance in trainingSet])
X = np.array([instance[1:] for instance in trainingSet])
clf1 = GaussianNB()
clf1.fit(X, Y)

from sklearn import tree
Y = np.array([instance[0] for instance in trainingSet])
X = np.array([instance[1:] for instance in trainingSet])
clf1 = tree.DecisionTreeClassifier()
clf1.fit(X, Y)
```

The average runtime and average accuracy is demonstrated like following:

```
decision tree average time consumed: 103.801600051
decision tree average accuracy: 6.72782874618
naive Bayes average time consumed: 13.60589993
naive Bayes average accuracy: 39.2660550459
sklearn Bayes average time consumed: 9.31220002174
sklearn Bayes average accuracy: 40.4892966361
sklearn decision tree average time consumed: 2.13099994659
sklearn decision tree average accuracy: 53.6391437309
```

From the figure 5, we can see that the rank for the accuracy is: sklearn_decision_tree > naive Bayes ≈ sklearn_Bayes >> decision tree. It is surprise to see the naive Bayes algorithm implemented with plain code has comparable performance with the sklearn_Bayes. We can see that the highest accuracy of sklearn_decision_tree method is around 53%, which is still not quite satisfactory.

The most critical observation is that there is an over-fitting problem for the decision tree implemented with plain code, which means the result can be affected by noise instance. While in the decision tree implemented using sklearn tool, the tree may has been post-pruned. For all those four methods, moderate fluctuation is observed, which should be related with the

random splited dataset.

From the figure 6, we can see that the rank for runtime is the following: sklearn_decision_tree sklearn_Bayes < naive Bayes << decision tree. It proves that the algorithm implemented with sklearn indeed runs faster, especially for the decision tree method. It is also observed that there is a strange trend for sklearn_Bayes method: the runtime is gradually increasing with the experiment number.

Building the code requires further attention to the details of those algorithms. From my experience of this project, I have learnt several other things that need to be keep in mind for data mining.

First, it is always important to preprocess the dataset as the first step. Some useless rows/columns in the dataset, which is provided as csv file, were deleted (e.g., the header in the first row and the IDs in the first column). Sometimes, the original dataset needs to be carefully examined before further steps were taken. Although this part is not in the core part in the algorithm, cleaning the data is always necessary to keep the following code concise, neat and easier for understanding.

Secondly, checking the code on a smaller partial dataset before applying it to the whole dataset is important too. This trick will help you check the correctness of the code easier and more efficiently.

Using the same dataset, the naive Bayes and decision tree algorithms was implemented using WEKA with 10 fold cross-validation. A zoom-in part of the decision tree is shown in figure7.

Table 1 Summary for naive Bayes classifier in WEKA

Time taken to build model: 0.23 seconds		
=== Stratified cross-validation ===		
=== Summary ===		
Correctly Classified Instances	957	96.6667 %
Incorrectly Classified Instances	33	3.3333 %
Kappa statistic	0.9663	
Mean absolute error	0.0007	
Root mean squared error	0.026	
Relative absolute error	3.4332 %	
Root relative squared error	26.0347 %	
Total Number of Instances	990	

Table 2 Summary for decision tree classifier (J48) in WEKA

Correctly Classified Instances	675	68.1818 %
Incorrectly Classified Instances	315	31.8182 %
Kappa statistic	0.6786	
Mean absolute error	0.0067	
Root mean squared error	0.0784	
Relative absolute error	33.48 %	
Root relative squared error	78.4306 %	
Total Number of Instances	990	

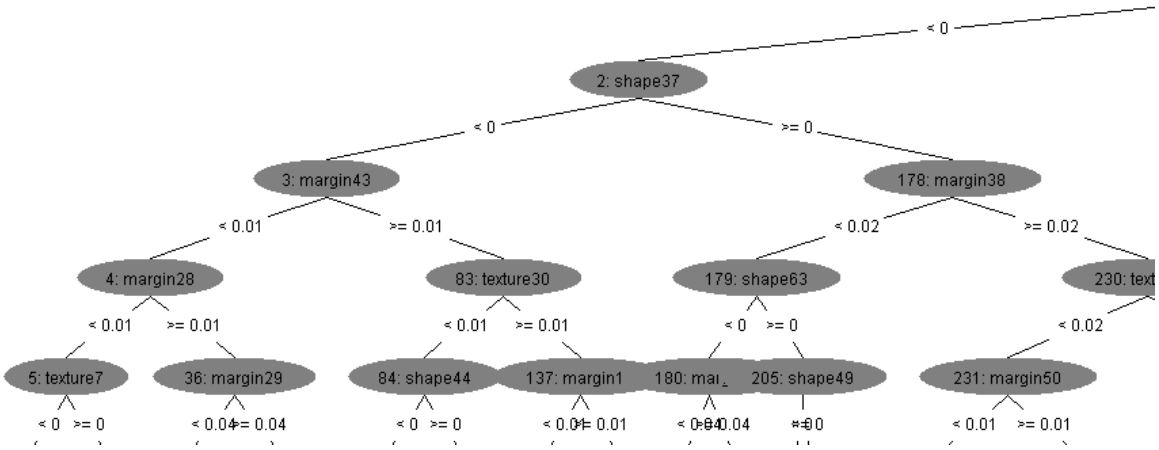


Figure 7: Partial zoom-in of the visualized decision tree

We can see that the naive Bayes classifier in WEKA has a very high accuracy (96.7%), while the decision tree has only 68.2% accuracy. Also, it is obvious that the model was built and validated (10 fold cross validation) in a short time compares with those implemented with python.