Clustering Algorithm and Its Application on Crime Data

Tianyuan Zhou

April 30, 2016

Cluster algorithms has many useful applications. K-means and its improvement, K-means++, are some of the most widely used clustering algorithm. By explaining the theory behind the method and apply it towards a real data set, we are able to gain better understanding of the algorithm, observe their strengths and shortcomings, and make a comparison between them.

1) Introduction

Cluster analysis, which is also known as data segmentation, is a very useful analysis that has a variety of goals. The main objective of the cluster analysis is to group a collection of objects into different subsets, which we called "clusters", such that the objects that are in the same cluster are more closely related to each other than objects that are in the different clusters. In other words, the objects in the same cluster will share the same characteristics, which can be both related to objects themselves or related to the relationships between the different objects. In the latter case, clustering algorithm can also used to form a hierarchical structure. In this structure, objects in the lower hierarchy will share part of the characteristics of objects in the upper hierarchy while clusters in the same hierarchy are more similar to each other than those in different groups.(2) Either way, the central role of clustering analysis is to study the similarities of different data points and find meaningful patterns between different groups of objects. (2) In my paper, I would focus on the first stated goal of cluster analysis and introduce K-Means related algorithm and applied it on the Boston Crime data set.

2) K-means algorithm

K-means, a term first coined by James McQueen in 1967 (5), is one of the most widely used clustering algorithms. Its aim is to solve the so-called "K-means problem". K-means problem, as defined in the paper by David Arthur and Sergei Vassilvitskii (1), involving a set of n data points where $X \subset$ ⍰^{d} and we applied the squared Euclidean distance

$$d(x_i, x_{i'}) = \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 = \|x_i - x_{i'}\|^2$$

as the dissimilarity method between different data vectors. Then, we want to find k number of points to serve as center (also called means) $C_1, C_2, \ldots, C_k$ such that their within-point scatter is minimized. The within-point scatter can be written as $W(C) = \sum_{k=1}^{K} N_k \sum \quad C(i) =$ $k\| x_i - \overline{x}_k\|^2$, where $\overline{x}_k$ is the mean vector (which is the center of the cluster) associated with the kth cluster and N_k is the sum of indicator function $I(C(i) = k)$. Therefore, the problem become solving for

$$C^{\ast} = \underset{\mathbf{C}} {\operatorname{arg\,min}}\sum_{k = 1}^{K}N_{k}\sum_{C(i) = k} \Vert x_{i}-\bar{x}_{k}\Vert^{2}$$

. Solving the whole problem exactly is NP-Hard (1) as the possible number of clusters increases exponentially which will quickly overwhelm any computers. However, in 1982, Stuart Lloyd proposed a local search method that proved to be fast enough. (4) His (and others) method take note that for any set of observation S we can obtained that $\bar{x}_{S} = \underset{m} {\operatorname{arg\,min}}\underset{i \in S}{\sum}\Vert x_{i}-m_{k}\Vert^{2}$, then we can obtain C^* by solving the enlarged optimization problem, which takes the same form as the initial problem but now minimized with respect to both C and $\{m_k\}_1^k$. In other words, we'll start with a given assignment, then iteratively minimize each cluster and change assignment until convergence. The full method, which is mentioned in the paper by Arthur and Vassilvitskii (1)(4), is shown below:

1) Arbitrarily choose a set of initial k centers $C = \{C_1, C_2, \dots, C_k\}$
2) For each $i \in \{1, \dots, k\}$, set the cluster C_i to be the set of points in X that are closer to c_i than they are to c_j for all $j \neq i$
3) For each $i \in \{1, \dots, k\}$, set c_i to be the center of mass of all points in C_i: $c_i = \frac{1}{|C_i|}\sum \quad x \in C_i x$
4) Keep repeating step 2 and 3 until the assignment C no longer changes.

In step 1, we usually choose the initial center uniformly at random in X. In step 2, if there are tie involved, we can break ties arbitrarily but we should stay consistent during the iterations in our method. In essence, the idea outlined above is trying to keep improving each individual cluster locally until it becomes impossible to do so. This algorithm converges very quickly in general, which is the reason that it is so popular. Algebra can shown (3) that Lloyd's algorithm's running time can be given as O(nkdi), where n is the number of observations, d is the dimension of data vectors, k is the number of clusters, and i is the number of iterations needed until convergence. However, there are several drawbacks for the K-means algorithm. One of the most important drawbacks is that the performance of K-means algorithm heavily depends on its initial choice of clusters. Because you are randomly choosing the initial assignment, there is no way you can make sure your randomly chosen point are actually spreading out. If your initial assignment themselves clustered together, or worse, if one of your initial assignment happened to fall on an outlier point, then it's very hard to get a good cluster result. This is especially true if your sample size is small and you have less points to choose from, then it is entirely possible that you grouped an outlier point as its own cluster even though in reality it fits in with one of the cluster that's already existed. This also shown outlier could cause problem in the K-means algorithm, especially if there are several of them. These drawbacks led other researchers try to improve on the idea. Some small improvements were made over the years, but the big breakthrough came in 2007 in the paper by Arthur and Vassilvitskii's paper(3) in 2007, where they developed a better and more efficient algorithm they called K-Means++.

3) K-means++ algorithm

As mentioned in the above section, the result and performance of K-means algorithm is heavily influenced by the choices of initial center, therefore the best way to try to improve the algorithm is trying to find the best initial center. Arthur and Vassilvitskii did exactly this. They proposed that since an ideal cluster should has minimum within-cluster distance and big between-cluster distance, then a good algorithm should pick points that are distant to each other. Their algorithm (1) is shown as below:

1) Take one center c_1, chosen uniformly at random from X, this part is as same as K-means.
2) Take a new center c_i, choosing $x \in X$ with probability $\frac{D(x)^2}{\sum \ x \in X D(x)^2}$
3) Repeat step 2, until we have taken k centers.
4) Now proceed as standard K-means algorithm.

In step 2, Arthur and Vassilvitskii applied what they called "D^2 weighting", which is the squared distance of a given point to c_1 over the sum of squared distance from every points to c_1. Since the denominator is constant, the only thing varies is the squared distance of each individual point. Thus, the closer a point to the initial center is, the less likely it will be chosen as the second center. This probability is decreased further as we are using the square of distances instead of absolute value. This step decreased the probability of selecting two points that are close to each other as center, and if your sample size is even marginally large, then this probability is essentially zero. The choosing of initial center takes some time, but once it was done, because it was already a very good initial assignment, the remaining algorithm runs very fast. In fact, as shown in their paper (1), if you let A be an arbitrary cluster from optimal cluster choice C^*, and C be the other random cluster, if we add random center to C from A, then $E[\phi(A)] < 8\phi^*(A)$, where ϕ is the loss function. Using this result, they then showed that

$$E[\phi(A)] \leq (\phi(A) + 8\phi^* - 8\phi^*(A)) \times (1 + H_{k-1})$$

, where H_{k-1} is the harmonic series to the k-1th term. As $H_{k-1} \leq 1 + \ln(K)$, this algorithm has complexity $O(\ln(K))$, this is faster than the regular K-means algorithm, which is treated as having linear complexity. Note that K-means++ algorithm does not address the concern that an outlier point can be assigned as the center of a new cluster. However, others have developed other way to combat this issue, in the form of that is similar to the "burn-in" of other numeric methods. They'll determine an n-start value, and only start the process after the nth randomly chosen starting point, this could decrease the chance of selecting an outlier as the starting point. The other method to mitigate the outlier issue is based on logic that an outlier would probably far enough from any point, thus we can exclude certain points that are far enough from all other data points. With these improvements, k-means++ algorithm in general gave better result than regular k-means algorithm, especially when there are outliers present. However, our discussion thus far on K-means and K-means++ both glossed over one crucial question: how to choose the number of k?

4) Choice of K

This question turns out to be remarkably complex, as the standard model selection techniques such as cross-validation-method is not very suitable for unsupervised learning problems such as clustering. The reason being that you are not sure what the cluster would look like before you applied the algorithm and therefore cannot ensure that you will be able to randomly divided data into training sets and testing sets. (2) In real world problem, sometimes you will have an idea how many cluster do you want. For example, it would be very reasonable to have number of clusters equal to number of districts if you are trying to run a cluster analysis on voting data. Otherwise, researchers developed a lot of method developed but there is no general consensus on which method is the best. One of the most widely used method is the so called "Elbow Method". (7) In this method, we usually take a look at either plot of within sum of squares or the plot of variance explained as a function of the number of clusters. At first the sum of squares will drop very quickly (or if you use variance explained, will rise very quickly), but after some point when you add in another cluster, the drop in sum of squares (or the rise in variance explained) become very slow, which will reflected on the graph as a sharp corner, looking like an elbow on the arm, which is how this method got its name. This method usually provide a k-value that lead to a K-means or K-means++ cluster that performs well. However, as this is a graphic-based method, sometimes it is hard to unambiguously see where the "Elbow Point" is.

One of the newer approach to this problem is the Jump method proposed by Catherine Sugar and Gareth James in 2003 (6), which determines the number of clusters that maximizes efficiency while minimizing error by information theoretic standards. (6) In this method, they let data set to be a random variable X that consists of a mixture distributions of some number of components with common covariance Γ. Then they show given cluster center $\{C_1, C_2, \dots, C_k\}$ with C_X be the center that is closest to a sample of given data set X, then the minimum average distortion per dimension when fitting the K centers to the data is shown to be

$$d_k = \frac{1}{p} \min_{c_1 \dots c_K} E[(X - c_X)^T \Gamma \quad -1(X - c_X)]$$

, whichever number of k that minimize this Mahalanobis Distance would be the optimal number of K to uses on the cluster. (6) This is still an area that a lot work can still be done and there could be new method to choose the optimal number of K developed in the near future.

5) Data Analysis

Next, let's apply K-Means algorithm and K-Means++ algorithm to a data set and compare them. For this analysis, we will use the Boston Crime Incident Report data set provided by Boston Police Department. The description of data sets claimed to be "all crime reported in the City of Boston" during a 3-year period. However, this description is not entirely accurate as the data also contains emergency response and other routine police works in it. The data contains all the police responses (whether it is a crime, infraction, fire, or medical emergency) during July 2012 to August 2015. The variables in the data set include a unique report number (COMPNOS), Nature Code, incident type description, crime code, reporting district, reporting area, reporting time, date, month, and year, weapon type, if there is shooting involved, if it is a domestic incident, shift, day of week, the street that incident occurs in, and the geographical coordinate that crimes appears in, there are 268056 observations. Our primary interested is in two of these variables: the geographical location and the type of crime.

a) Data Cleaning and transforming

Before conducting the clustering algorithm, we need to clean the data a little bit. First of all, as the data set including all the police responses, there are quite few categories in crime type that does not correspond to an actual crime (e.g. medical assistance, missing person located) or police works that you don't usually considered as a crime happened (e.g. towed, Search Warrant). Some of those categories such as medical assistance actually takes up quite a lot portion (nearly 1/20) of the total responses that include them will severely skewed our analysis on crime clusters, therefore I decided to remove them. There are still some variable left that its meaning or scope are ambiguous. For example, VAL, which stands for violation of auto laws, range from minor infractions such as speeding to serious offenses such as DUI or driving without a license, which is very ambiguous on whether it is crime-related, therefore I decided to keep it. Another problem with the data set is that there are a few incidents in our data set that has longitude, latitude, or both with value 0. As Boston neither lies on equator nor on the Greenwich Time Line, we can remove these points as well since we wish to cluster on the geographical data. After the removal of these data points, we have 217691 observations, still plenty enough to run the cluster algorithms.

Next, because the K-means and K-means++ algorithm only works with quantitative type data but the variable Crime Incident Description is, as its name suggests, in words. Therefore, we need transform this variable into numbers. One way we could do that is to use a dummy variable representation for each incident category. However, some of the crimes are similar. For example, "simple assault" and "aggravated assault" are both assaults, albeit with different level of severity. "Residential Burglary" and "Commercial Burglary" are both burglaries, albeit on different subjects. If we want to capture these similarities, we should not use a simple dummy variable approach. R has several packages such as tm or lda that specialized in vectorizing string objects. However, for this project, I decided to use Python and utilize TfidfVectorizer in the sklearn module. This object is a combination of CountVectorizer and TfidfTransformer in Python. CountVectorizer make a list of all unique words appears in the data frame and return a list of list that contains the number of time that each word appears in each vectorizer. Then, TfidfTransformer transform the data using term-frequency times inverse document-frequency by formula $\mathrm{tf} * (\mathrm{idf} + 1)$, this will turn the words in "Type" variable into a data vector, which then I can store as a matrix and passed in the K-means algorithm. As for longitude and latitude, they were already in the numeric form, so I just ran "scale" command to standardize them.

b) Choosing K

I will be using Elbow method to get a rough idea about the optimal number of k. In python, I created a list to store the number of k and another list to store the corresponding error value, which in python is defined as the sum of squares of distances of samples to their closest cluster center. Then I run the K-means and K-means++ algorithm on each one and appended error value into the list. My error plot looks like the following:

Error by Number of Clusters
From the graph above, it appears that if there is an "elbow" present in the graph, it'll be somewhere around 4 or 5. However, I do have some prior knowledge about the city of Boston and know there are more than 4 or 5 neighborhoods present, so I probably need a few more clusters than what the elbow method suggests or K-means algorithm would just grouped everything into 4 or 5 general areas, which is not what I want. I decided to use k = 12 as this is the first k-value that if you add in one more k, the error might actually increases.

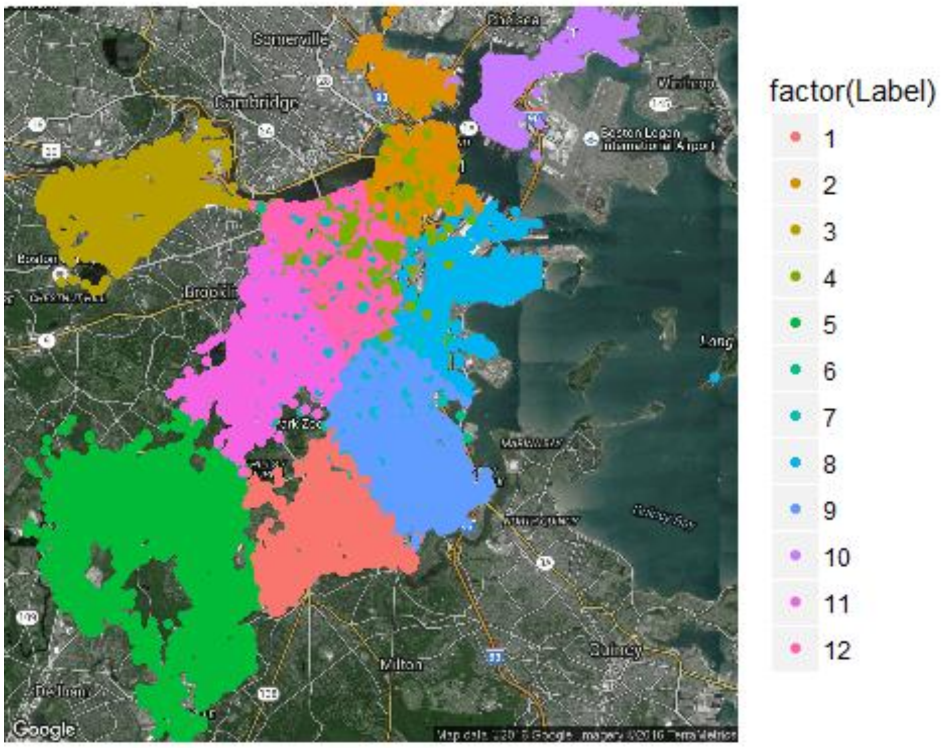
(c)K-means cluster

After we decided the number of clusters to use, we could began to run our cluster analysis. For K-means, we'll use the "kmeans" command in R. The default setting of the "kmeans" command in R will choosing the initial center randomly, and then update the cluster assignment based on Lloyd's method outlined in Section (2). Its default nstart value is 1, which means there are no "burn-in"s and we will use the first center we have chosen. The default iter.max value, which set the maximum number of iterations allowed, is 10. The Code runs fairly quickly and we get a resulting cluster assignment. I would then plot the clusters using "ggmap" packages. (3) First, I passed in the list of longitude and latitude data of all the incident in the data set into "make_bbox"" function, this will return to be an optimal scope to extract my map. Then I use "qmap" function to get the map that zoom in the box I just made. Finally, I superimposed my data point on the map and color each cluster differently, the resulting plot is shown below:

Judging from the plot, all the police response incidents has been divided into 12 parts roughly based on their geography location, with the crime in the same neighborhood tends to be in the same cluster. However, as we are interested in the type of crime as well as the location we can go into each cluster and extract this information. I used R package "sqldf" to implement some SQL command on the data to extract what is the most prevalent crime/violation in this cluster, the result is shown below:

```
#I'm required to show these codes due to Google API User Agreement.
mapBox <- make_bbox(crime$longitude, crime$latitude)
BostonMap <- qmap(mapBox, zoom = 12, maptype = "hybrid", legend = "none")

## Warning: bounding box given to google - spatial extent only approximate.

## converting bounding box to center/zoom specification. (experimental)

## Map from URL :
http://maps.googleapis.com/maps/api/staticmap?center=42.313347,-
```

```
71.074948&zoom=12&size=640x640&scale=2&maptype=hybrid&language=en-
EN&sensor=false

BostonMap +
  geom_point(aes(x = crime$longitude, y = crime$latitude, colour =
factor(Label)),
             data = crime)

## Warning: Removed 2 rows containing missing values (geom_point).
```



Cluster 1 (Codman Square), 2 (Downtown Boston), and 3 (Allston and Fenway) are areas with immense amount of traffics. Thus, unsurprisingly, auto law violations tends to be the most frequent infractions. Cluster 8, which has much less traffic congestion (due to the highway around it) but near a lot of parks and college campus, has drug related charges as the leading infraction. (In case you are wondering, drug related charges are high in cluster 3 as well, just not as much as auto law violations) Clusters 11 and 12, which is in residential area, has Larceny as the main crime.

```
##    Label                         Type MAX(t.count)
## 1      1                          VAL         2024
## 2      2                          VAL         3277
## 3      3                          VAL         2153
## 4      4                OTHER LARCENY        14327
## 5      5                          VAL         2674
## 6      6                          VAL        12280
## 7      7               SIMPLE ASSAULT         8451
## 8      8                 DRUG CHARGES         1091
## 9      9                  VANDALISM         2898
## 10    10                        VAL         1321
## 11    11              OTHER LARCENY         2013
## 12    12 LARCENY FROM MOTOR VEHICLE         3464
```

One good thing to see is there are some spot in Bunker Hill area, which does not directly connected to East Boston area (cluster 10), being grouped together with cluster 10 instead of nearby cluster 2. The leading crime for both clusters is VAL, but that purple spot is near the Bunker Hill Monument, where vandalism is prevalent, while the most prevalent crime in Bunker Hill community outside of the monument area is assault. It is good to see that our k-means algorithm did capture this information and assigned clusters accordingly instead of just relied on geographical location. Also, there is an outlying point on Long Island, but it didn't get randomly chosen as the center of a cluster, so we are fine.

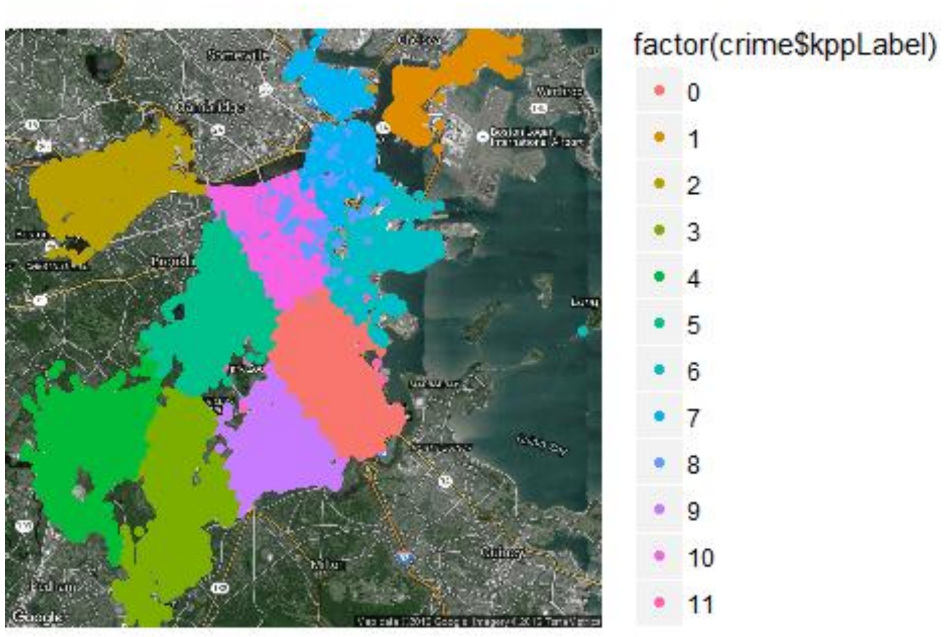
```
cluster2crime <- sqldf("SELECT Type, COUNT(Type)
                       FROM crime
                       WHERE Label = 2 GROUP BY Type
                       ORDER BY COUNT(Type) DESC LIMIT 2;")

cluster10crime <- sqldf("SELECT Type, COUNT(Type)
                        FROM crime WHERE Label = 10
                        GROUP BY Type
                        ORDER BY COUNT(Type) DESC LIMIT 2;")

cluster2crime

##             Type COUNT(Type)
## 1            VAL        3277
## 2 SIMPLE ASSAULT        2347

cluster10crime

##        Type COUNT(Type)
## 1       VAL        1321
## 2 VANDALISM         870
```

(d)K-means++ cluster

Now we implement K-means++ algorithm. In R there is a package "LICORS" that has a build in "kmeanspp"" function that can run K-means++ for me. However, probably due to some inefficient coding, that code runs very slow, contrary to what claimed in the Arthur and Vassilvitskii paper. Therefore, I used Python to implement the algorithm instead. Just like in kmeans, I set max.iter to 10, and nstart to 1. After I get the cluster assignment in Python, I passed the assignment in R and again use ggmap (3) to plot the cluster with the map, and the result is shown below:

```
BostonMap +
  geom_point(aes(x = crime$longitude, y = crime$latitude, colour =
factor(crime$kppLabel)),
             data = crime)
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



A cursory glance shows that a lot of data are be clustered exactly the same way, particularly the cluster in Logan Airport Area, the Allston and Fenway area, and the UMASS campus area all have no visible difference than the K-means cluster from the previous step. However, there are some differences. For instance, the entire Hyde Park area was represented by a single cluster (cluster5) in the K-means algorithm, but now K-means++ algorithm now break them into two clusters. (cluster 3 and cluster 4) You could make an argument that this assignment is better as cluster 3 is near Mattapan, which is generally not considered a safe neighborhood, assault and larceny is a common crime here. While cluster 4, which is near Roslindale, is considered a much safer place, and the most common crime or infraction over there is drug related. Thus, what kind of crime you expect to see in Hyde Park neighborhood really depends on which part of Hyde Park you lived in. If we ran the same kind of SQL command as we did in K-means cluster, you can see this information is portrayed by the K-means++ assignment: cluster 3 and cluster 4 seems to span around same area, but cluster 4 has both a lot less incident happens(top 4 sum to 2547 compare to 5284) and less incident that would be considered a crime. (In cluster 4, only drug charges is an event that's unambiguously clear that a crime has been committed)

```
##    kppLabel                      Type MAX(t.count)
## 1         0            SIMPLE ASSAULT         3287
## 2         1                       VAL         1323
## 3         2                       VAL         2153
## 4         3                       VAL         2203
## 5         4                       VAL          775
## 6         5             OTHER LARCENY         1981
## 7          6             SIMPLE ASSAULT         1135
## 8          7                        VAL         3691
## 9          8              OTHER LARCENY        14352
## 10         9             SIMPLE ASSAULT         1777
## 11        10 LARCENY FROM MOTOR VEHICLE         3334
## 12        11                        VAL        11539
```

```
t2cluster3crime <- sqldf("SELECT Type, COUNT(Type)
                         FROM crime
                         WHERE kppLabel = 3
                         GROUP BY Type
                         ORDER BY COUNT(Type) DESC limit 4;")

t2cluster4crime <- sqldf("SELECT Type, COUNT(Type)
                         FROM crime
                         WHERE kppLabel = 4
                         GROUP BY Type
                         ORDER BY COUNT(Type) DESC limit 4;")
t2cluster3crime
```

```
##             Type COUNT(Type)
## 1            VAL        2203
## 2  OTHER LARCENY        1063
## 3          MVAcc        1013
## 4 SIMPLE ASSAULT        1005
```

```
t2cluster4crime
```

```
##            Type COUNT(Type)
## 1           VAL         775
## 2 DRUG CHARGES          625
## 3         MVAcc         621
## 4        InvPer         526
```

(e) Comparisons

However, eyeballing and using prior knowledge is not a rigorous way to decide which cluster assignment is better. Therefore, we should refer back to our previous goal that we want to minimize the Within-Sum-of-Squares. For this particular iteration, our K-means cluster assignment has a within-sum-of-squares of 228424.3 while my K-means++ algorithm has a within-sum-of-squares of 224985.3, which is slightly smaller. However, those values are very similar and in practice, both are reasonable given that we have 217691 data points to be clustered. However, as the initial assignment of K-means and K-means++ can be different each time we run the algorithm, their resulting error would be different each time as well, thus it would be better to run both method a few times and take the average. The resulting plot is shown below:

Comparisons of error between K-means and K-means++ algorithm
As shown above, we can see that two lines are very similar, to the degree that it's almost identical. One possible reason is that our sample size is really large, so it is very unlikely

that our initial K-means assignment would assign centers that are very close to each other or assign an outlier as center, especially since we only have one point that would be considered an outlier. If our sample size is much smaller, than the effect of different initial assignment might be more obvious and will be more reflected upon by the error value. With that being said, K-means++ algorithm is still better than K-means algorithm as the red lines (k-means) stay consistently above the blue line (K-means++), which support the notion that K-means++ would generate better clusters.

6) Conclusion

Based on the results from our analysis in Boston Crime Incident data set, K-means++ does seem to give better cluster assignment than the K-means algorithm. In our data analysis, K-means++ captured more information regarding the type of crime committed as well as geographical location than the standard K-means algorithm, it also results in a cluster that has less within-sum-of-squares than the K-means algorithm. However, it seems that at large sample, the performances of two clusters are nearly indistinguishable because the probability of randomly chose a "wrong" point is very small. In that case, although K-means++ is slightly more accurate and runs faster once the initial assignment is chosen, the cost of waiting for it to choose the initial assignment might make kmeans algorithm cheaper and more appealing. The advantages of K-means++ can be more directly observed if either your data set is really small that there is a realistic chance to select a "wrong" starting point for K-means algorithm or if the data set is large enough that the advantage of quicker processing time outweighs the cost of waiting for it to come up with a good initial assignment.

References

[1]Arthur, D. and Vassilvitskii, S. (2007). "k-means++: the advantages of careful seeding" (PDF). *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics Philadelphia, PA, USA.* pp. 1027-1035.

[2]Hastie, Trevor, Tibshirani, Robert, and Jerome Friedman. "Unsupervised Learnings." *The Elements of Statistical Learning.* 2nd ed. Vol. 1. New York: Springer, 2009. 509-11. Print.

[3] D. Kahle and H. Wickham. ggmap: Spatial Visualization with ggplot2. *The R Journal*, 5(1), 144-161.

[4]Lloyd., S. P. (1982). "Least squares quantization in PCM" (PDF). *IEEE Transactions on Information Theory* 28 (2): 129-137.

[5]MacQueen, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press.* pp. 281-297.

[6]Sugar, Caterine A. and Gareth M. James (2003). "Finding the number of clusters in a data set: An information theoretic approach". *Journal of the American Statistical Association* 98 (January): 750-763.

[7]Thorndike, Robert L. (December 1953). "Who Belongs in the Family?". *Psychometrika* 18 (4): 267-276

Appendix

Initial preparation code

```
# setwd("C:/Tianyuan/Stats/MA 751/Project")
# library(stringr)
# crime <- read.csv("Crime_Incident_Reports.csv")
# names(crime)[3] <- "Type"
# crime <- crime[-which(crime$Type == "TOWED"),]
# crime <- crime[-which(crime$Type == "MedAssist"),]
# crime <- crime[-which(crime$Type == "Medical Assistance"),]
# crime <- crime[-which(crime$Type == "Missing Person Located"),]
# crime <- crime[-which(crime$Type == "Service"),]
# crime <- crime[-which(crime$Type == "Arrest"),]
# crime <- crime[-which(crime$Type == "SearchWarr"),]
# crime <- crime[-which(crime$Type == "PhoneCalls"),]
# crime <- crime[-which(crime$Type == "DEATH INVESTIGATION"),]
# crime <- crime[-which(crime$Type == "Motor Vehicle Accident Response"),]
# crime <- crime[-which(crime$Type == "Police Service Incidents"),]
# crime <- crime[-which(crime$Type == "Investigate Person"),]
# crime <- crime[-which(crime$Type == "Warrant Arrests"),]
# crime <- crime[,-c(1, 2, 4, 5, 6, 11, 12, 15, 16, 17, 18, 19)]
# longitude <- c()
# latitude <- c()
# crime$Location <- sapply(crime$Location, toString)
# for(i in 1:length(crime$Location)){
#   latitude[i] <- as.double(substring(strsplit(crime$Location[i],
",")[[1]][1],
#      2, nchar(strsplit(crime$Location[i], ",")[[1]][1])))
#   longitude[i] <- as.double(substring(strsplit(crime$Location[i],
",")[[1]][2],
#                    2, nchar(strsplit(crime$Location[i], ",")[[1]][2])-1))
# }
# crime$longitude <- longitude
# crime$latitude <- latitude
# crime <- crime[,-8]
# write.table(crime, file = "C:/Tianyuan/Stats/MA
751/Project/BostonCrimes.csv", sep = ",",
#              row.names = FALSE)
```

Python code for vectorizing crime type data

```
# crimedf = pd.read_csv('C:/Tianyuan/Stats/MA 751/Project/BostonCrimes2.csv')
# crimeType = []
# for i in range(len(crimedf)):
#     if crimedf["longitude"][i] != 0:
#         crimeType.append(crimedf["Type"][i])
# vec = TfidfVectorizer(min_df=1)
# crimev = vec.fit_transform(crimeType).toarray()
# crimevdf = pd.DataFrame(crimev)
# crimevdf.to_csv('C:/Tianyuan/Stats/MA 751/Project/BostonCrimes.csv', index
= False)
```

Python code for choosing K

```
# def evaluate_clusters2(min_clusters, max_clusters, X):
#     for k in range(min_clusters, max_clusters+1):
#         knum.append(k)
#         kmeans = KMeans(init = 'k-means++', n_clusters = k, n_init = 3,
max_iter = 20)
#         kmeans.fit_predict(X)
#         error.append(kmeans.inertia_)
#
#     plt.plot(knum, error)
#     plt.xlabel('Number of clusters')
#     plt.ylabel('Error')
#
# knum = []
# error = []
# evaluate_clusters2(1, 30, temp2)
```

K-means algorithm implementation and analysis

```
# suppressMessages(library(ggmap))
# suppressMessages(library(ggplot2))
# suppressMessages(library(sqldf))
# cl <- colors()
# setwd("C:/Tianyuan/Stats/MA 751/Project")
# crime <- read.csv("BostonCrimes.csv")
# crime <- crime[-which(crime$longitude==0),]
# crime$newLong <- scale(crime$longitude, TRUE, TRUE)
# crime$newLat <- scale(crime$latitude, TRUE, TRUE)
# crimeTypeVector <- read.csv("CrimeVector.csv")
# input <- data.frame(crime$newLong, crime$newLat, crimeTypeVector)
# Label <- kmeans(input, 12)[1]$cluster
# crime$Label <- Label
# crime$kmeansColor <- cl[crime$Label+10]
# mapBox <- make_bbox(crime$longitude, crime$latitude)
# BostonMap <- qmap(mapBox, zoom = 12, maptype = "hybrid", legend = "none")
# BostonMap +
#   geom_point(aes(x = crime$longitude, y = crime$latitude, colour =
factor(Label)),
#              data = crime)
# suppressMessages(t1 <- sqldf("SELECT t.Label, t.Type, MAX(t.count) FROM
(SELECT Label, kmeansColor, Type, COUNT(Type) as count FROM crime GROUP BY
Label, Type) as t GROUP BY t.Label"))
# t1
# cluster2crime <- sqldf("SELECT Type, COUNT(Type)
#                         FROM crime
#                         WHERE Label = 2 GROUP BY Type ORDER BY COUNT(Type)
DESC LIMIT 2;")
#
# cluster10crime <- sqldf("SELECT Type, COUNT(Type)
#                          FROM crime WHERE Label = 10
#                          GROUP BY Type ORDER BY COUNT(Type) DESC LIMIT 2;")
#
# cluster2crime
# cluster10crime
```

Python implementation of K-means++ algorithm

```
# ss = StandardScaler()
#
# newLong = ss.fit_transform(longitude)
# newLat = ss.fit_transform(latitude)
#
# temp2 = np.vstack((newLong, newLat, crimev.T)).T
#
# ncluster = 12
# kmeans = KMeans(init='k-means++', n_clusters=ncluster, n_init=1)
# kmeans.fit_predict(temp2)
# centroids = kmeans.cluster_centers_
# labels = kmeans.labels_
# error = kmeans.inertia_
#
#
# print "The total error of the clustering is: ", error
# print '\nCluster labels'
# print labels
# print '\n Cluster Centroids'
# print centroids
#
# kppLabels = pd.DataFrame({"kpplabel": labels})
# kppLabels.to_csv('C:/Tianyuan/Stats/MA 751/Project/kppLabels.csv', index =
False)
```

K-means++ related analysis

```
# kppresult <- read.csv("kppLabels.csv")
# crime$kppLabel <- kppresult$kpplabel
# crime$kppColor <- cl[crime$kppLabel + 10]
# BostonMap +
#   geom_point(aes(x = crime$longitude, y = crime$latitude, colour =
factor(crime$kppLabel)),
#               data = crime)
# t2 <- sqldf("SELECT t.kppLabel, t.Type, MAX(t.count) FROM (SELECT kppLabel,
kppColor, Type, COUNT(Type) as count FROM crime GROUP BY kppLabel, Type) as t
GROUP BY t.kppLabel")
# t2
# t2cluster3crime <- sqldf("SELECT Type, COUNT(Type)
#                           FROM crime
#                           WHERE kppLabel = 3
#                           GROUP BY Type
#                           ORDER BY COUNT(Type) DESC limit 4;")
#
# t2cluster4crime <- sqldf("SELECT Type, COUNT(Type)
#                           FROM crime
#                           WHERE kppLabel = 4 GROUP BY Type
#                           ORDER BY COUNT(Type) DESC limit 4;")
# t2cluster3crime
# t2cluster4crime
```

code for the error comparison plot

```
# kmeansError <- c()
# for(i in 1:30){
#   kmeansError[i] <- kmeans(input, i)[5]$tot.withinss
# }
# kmerror <- data.frame(error$k, kmeansError)
# names(kmerror)[1] <- "k"
# names(kmerror)[2] <- "error"
# kmerror$type <- "K-means"
# error <- read.csv("algorithmError.csv")
# names(error)[1] <- "k"
# names(error)[2] <- "error"
# error$type <- "K-means++"
# algorithmError <- rbind(kmerror, error)
# ggplot(algorithmError, aes(x = k, y = error, color = type)) +
#   geom_line(size = 1) + xlab("Number of Clusters") + ylab("Error") +
#   ggtitle("Comparison Between K-means and K-means++") +
#   theme(title=element_text(size=15))
```