**Soft K-Means Clustering:**

**Theory and Application with Uber Pickup Data from NYC**

Evan Generoli

Statistical and Machine Learning

Professor Hendricks

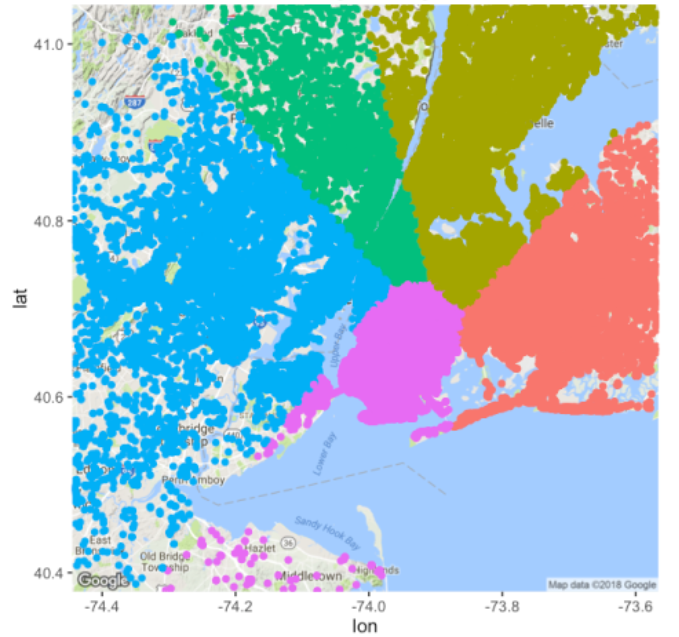
Fall Semester 2018

University of Colorado Denver

**Introduction**

As interest in the field of statistical/machine learning has exploded with the data revolution in recent years, many previously obscure areas of the discipline such as unsupervised learning have come to forefront of desired skills in the modern economy. As more companies realized that techniques like clustering, which partition a dataset into mutually exclusive groups of similar observations, can be used for applications like market segmentation and insurance risk analysis, these techniques have become highly sought after.

One algorithm commonly used for such clustering applications is the k-means algorithm. A good example of the k-means algorithm was presented on DataCamp, a data science education website. In the example, author Sejal Jaiswal, shows how latitude and longitude data of Uber pickups in New York City can be clustered into five groups that match up relatively well with the city’s five boroughs (Jaiswal, 2018). While this is a fine example of the k-means algorithm, this algorithm is inadequate to address the issue of what to do about places along the borders between clusters/boroughs. The mutually exclusive nature of clusters, ie. an observation belonging to one cluster implies it does *not* belong to another cluster, means that border points are assigned only to one cluster or another, which seems less than ideal. The different neighborhoods, business districts, etc. within NYC (and their associated subcultures) do not have hard cutoffs along borough borders, but rather gradually fade in/out of one another. In order to deal with this issue, a new paradigm is required.



**Figure 1**: NYC Uber pickups clustered with classical k-means

*Figure taken from Jaiswal, 2018*

Enter soft k-means clustering. The soft aspect here refers to the fact that under this algorithm, the assignment of datapoints to clusters is no longer “hard.” The classical (hard) k-means algorithm imposes that membership of datapoints in clusters is binary, ie. a datapoint either belongs to a given cluster or doesn’t. On the contrary, the soft k-means algorithm allows every datapoint to have a *degree* of membership in each cluster. This is computed such that these “degrees of membership” sum to one for each observation, meaning that they can be interpreted as probabilities. This means that the probability associated with observation and cluster can be thought of as the probability that observation belongs in cluster . This model of clustering then allows for a more intuitive solution to the problem of datapoints situated on the border between two or more clusters for which it is not obvious to which cluster they should be assigned. This algorithm should then be better equipped to handle such situations as the aforementioned border points between NYC boroughs.

**Case Study**

Therefore, as an illustration of soft clustering, I apply the soft clustering to the above-mentioned dataset on Uber pickups in New York City, freely available on Kaggle. The original dataset contains information on 4,534,327 pickups in NYC from April through September of 2014. The raw data only contained four variables: latitude of pickup, longitude of pickup, date & time of pickup, and the NYC Taxi & Limousine commission base company code associated with the pickup. Table 1 presents the summary statistics from the dataset. For interpretability, the date/time information has been recoded as separate variables. Note that the the NYC Taxi & Limousine commission base company code variable did not appear useful, and was removed.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Table 1: Raw Summary Statistics** | | | | | | |
|  | Mean | Std. Dev. | Median | Min | Max | N |
| Lat | 40.74 | 0.04 | 40.74 | 39.66 | 42.12 | 4534327 |
| Lon | -73.97 | 0.06 | -73.98 | -74.93 | -72.07 | 4534327 |
| Month | 6.83 | 1.7 | 7 | 4 | 9 | 4534327 |
| Day | 15.94 | 8.74 | 16 | 1 | 31 | 4534327 |
| Weekday | 4.21 | 1.91 | 4 | 1 | 7 | 4534327 |
| Hour | 14.22 | 5.96 | 15 | 0 | 23 | 4534327 |
| Minute | 29.4 | 17.32 | 29 | 0 | 59 | 4534327 |

Unfortunately, this dataset proved to be too large. For reasons of computational feasibility, I randomly sample 30,000 observations from the dataset. Additionally, for simplicity I removed the date/time information, leaving only the latitude and longitude variables. Table 2 presents summary statistics for the truncated data.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Table 2: Final Dataset Summary Statistics** | | | | | | |
|  | Mean | Sd | Median | Min | Max | N |
| Lat | 40.74 | 0.04 | 40.74 | 40.11 | 41.28 | 30000 |
| Lon | -73.97 | 0.06 | -73.98 | -74.65 | -72.74 | 30000 |

The goal of the cluster analysis in this case study is to cluster observations into five fuzzy clusters and plot on a map of NYC to visualize the soft nature of the bordering areas between clusters, ie. how the clusters fade into one another, in direct contrast to the hard boundaries observed with the classical k-means algorithm.

**Model & Methods**

In order to explain the mathematics of the soft k-means algorithm, a review of the classical (hard) k-means algorithm is in order. Note that my explanation of the following theory largely follows the structure and mathematics of lecture notes from Christian Bauckhage of the Univeristy of Bonn (Bauckhage, 2015a & 2015b).

Recall that the basic objective of k-means clustering is to partition a set

of datapoints into clusters of similar (as defined by squared Euclidean distance) observations. More formally, this means that each cluster is a proper subset of , such that

Additionally, the disjoint and exhaustive nature of these cluster sets tells us that each and every datapoint belongs to one cluster and one cluster only, ie. .

In order to group the most similar (again, by squared Euclidean distance) observations together and satisfy these conditions, standard k-means clustering can be thought of as the problem of choosing cluster centroids (means of datapoints in a cluster) such that the scatter or within cluster sum of squares (variance using squared distance) is minimized. Formally, this means that the cluster centroids are chosen to minimize the loss function

Unfortunately, finding a global minimizer for this loss function proves to be quite difficult computationally. This problem is classified as NP-Hard (very difficult) within computational complexity theory (Bauckhage, 2015a, p. 2), meaning that for most applications it is not computationally feasible to find the global optimum.

This is where the k-means algorithm comes in. Given that finding the global optimum is generally infeasible, the k-means algorithm utilizes greedy iterative descent to find a local minimum of the loss function. The algorithm starts by randomly choosing the cluster centroids . Given these initial centroids, each datapoint is then assigned to the cluster with the nearest centroid (by squared Euclidean distance). Formally, the algorithm is partitioning the set into clusters where

After all of the clusters have been delineated, the algorithm recomputes each cluster centroid as the mean of all datapoints which now belong to the cluster, ie. for each cluster, the algorithm computes

This process of determining the clusters and computing the means is then repeated until no datapoints change clusters. This is the classical (hard) k-means algorithm. The key assumption being imposed here is that each datapoint belongs to only one cluster. However, as mentioned, this may not always be optimal. There are often datapoints on the “border” between two (or more) clusters for which it is not clear to which cluster they belong.

Soft k-means clustering (sometimes fuzzy or weighted k-means) relaxes this assumption and allows datapoints to have a degree of membership to all clusters. This is done such that the degrees are interpretable as probabilities, ie. for a given observation the cluster membership degrees are between zero and one, and sum to one. Formally, each observation is assigned a degree of membership to each cluster, such that , which makes these proper probabilities

The easiest way to formalize this is by relating it to the hard k-means paradigm. Recall that under standard k-means, the loss/objective function to minimize was

which can be written as

where are indicator variables denoting whether observation is an element of cluster

Recalling that the k-means algorithm, given a set of centroids , assigns each observation to the cluster with the nearest centroid, the can be written as

Here we can see that the cluster membership indicators are based on distance, and are strictly binary, ie. . This is a result of the hard cluster assignments under the classical k-means algorithm. However, under soft k-means, instead of binary cluster membership indicators, we want probabilities of cluster membership for each observation-cluster pair. That is, we want a “continuous membership indicator” between zero and one () that describes the *degree* to which observation belongs to cluster based on the distance from datapoint to centroid . As mentioned, in order to be properly considered probabilities, the “continuous membership indicators” need to sum to one for each observation (), as the probability of observation belonging to *any* of the clusters is one (or equivalently, the probability of belonging to none of the clusters is zero).

With this understanding in mind, the issue becomes that of achieving a transformation from squared Euclidean distances into probabilities , given a set of centroids . One such transformation that is commonly used is the softmax or normalized exponential function. The softmax function takes a vector and normalizes its elements into a probability distribution. In this context the vector to be normalized is the length vector of distances to the cluster centroids for a given observation . Formally, the probabilities are computed as

where is the stiffness parameter which controls the degree to which the clusters are soft/fuzzy vs. crisp/hard. We can see that the numerator returns a number between zero and one for each element of the distance vector, and the denominator scales this number such that the elements of the distance vector sum to one. With this in mind, it is not hard to see that the objective function to be minimized in the soft k-means context is very similar to that of the hard k-means written with cluster indicator variables. The key difference here is that the binary (hard) cluster indicators have been replaced with probabilities (or “soft/continuous membership indicators”) . Formally, soft k-means clustering seeks to choose cluster centroids which minimize the loss function

where the are the cluster membership probabilities

With this understanding in mind, it is time to move on the centroid updating step of the algorithm. A relatively easy way of understanding this issue is to examine a single cluster with centroid . Note that this particular cluster’s portion of the loss function is

If we could choose each such that each corresponding is minimized, we would minimize the overall loss function. Since the centroid update step of the soft k-means algorithm takes the membership probabilities as given, and is a convex function in , the optimum can be found by simply examining the first order conditions for minimization. First, note that

Now, taking the derivative of with respect to and setting equal to zero

And finally, solving for

That is, in order to minimize each cluster’s contribution to the overall loss function, we set each to the weighted mean of all datapoints , where the weights are membership probabilities for cluster and each observation , and the “n” to be divided by is the sum of all the membership indicators for the given cluster.

Given these methods for calculating cluster membership probabilities and updating cluster centroids, it is not hard to see how the classical (hard) k-means algorithm is tailored to fit the soft clustering paradigm. As in the classical algorithm, the first step is to randomly initialize the cluster centroids. Second, the algorithm calculates all the cluster membership probabilities according to

Third, the algorithm updates each cluster centroid to the weighted mean of all datapoints

The second and third steps are then iterated until the cluster centroids stabilize.

Now, to be clear, this particular formulation of the soft k-means algorithm assumes that the softmax function is the “correct” probability mapping function to use. This means that we are assuming that the exponential function in the numerator of the softmax function is the proper shape of probability distribution to map squared Euclidean distances to membership degrees (note that the sum in the denominator is merely a standardizing constant). This is not necessarily always ideal. For instance, the variant of soft/fuzzy clustering known as Fuzzy C-means (FCM) uses a different mapping function, which makes a different distributional assumption. Under FCM, the cluster membership probabilities are computed as

Where is the “fuzzifier” (this corresponds to the stiffness parameter from the softmax formulation). The “best” choice of probability mapping function, if such a thing exists, requires exercising discretion with regard to the context.

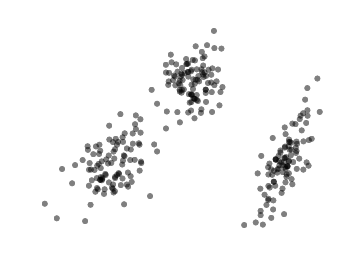
Unfortunately, soft clustering cannot handle missing data. As we have seen, the method hinges on the calculation of Euclidean distances. If there are any missing elements in a given datapoint vector , the algorithm will be unable to compute the Euclidean distance from to other datapoints. Therefore, any observations with missing data must be handled in typical ways (dropping, imputation, etc.) prior to implementing the algorithm.

Given this reliance on distances, it seems relatively clear that quantitative data is required for implementing soft clustering. Distances cannot be calculated with ordinal or categorical variables directly without imposing certain unrealistic assumptions, ie. every jump from one value to the next (or one category to the next) should be weighted the same. However, these issues can be sidestepped by transforming ordinal or categorical variables into quantitative variables. With regard to categorical variables, the solution is to simply dummy code them, ie. replace the variable with dummy (binary) variables for each category of said variable. For ordinal variables, they must be transformed into quantitative measures by functions such as

as was suggested in class.

As with the classical k-means clustering algorithm, the choice of whether to standardize variables before clustering will depend on context. Standardizing the variables will likely change the clusters computed, so the decision to standardize will need to be based on whether the researcher/statistician/etc. thinks that a particular variable with a greater variance should drive the distance measure more. That being said, data should be standardized at least in the sense that variables should be on the same scale, ie. we don’t want one variable measured in miles and another in kilometers.

In sum, the soft k-means algorithm offers more flexibility than the classical (hard) k-means algorithm by adapting k-means to contexts where hard cluster assignments do not seem to make sense. These contexts seem to be relatively common, as it doesn’t always seem natural to think there are clear delineations between groups. Therefore, the soft k-means algorithm should be used when it is not clear where one group ends and another begins. With regard to market segmentation in advertising for instance, if certain individuals are in the middle of two or more apparent groups identified for targeted advertising, it seems to make more sense to assign these individuals a degree (probability) of membership to multiple groups than to impose that they must only belong to one group. One can then decide what the proper cutoff (in terms of probability) is for receiving a particular advertisement, and maybe an individual will receive more than one type of advertisement.



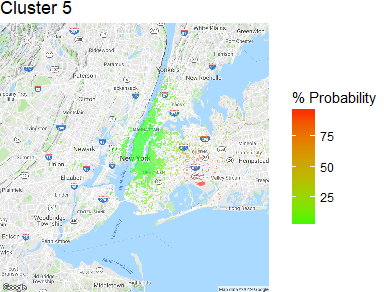
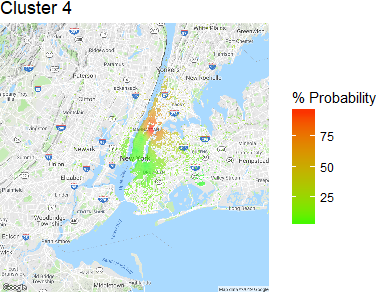
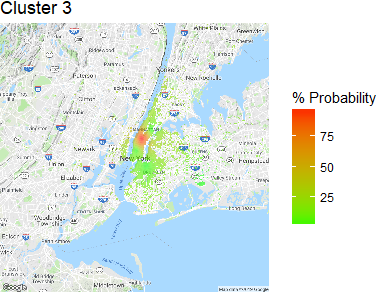
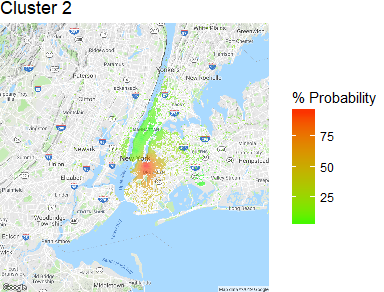
**Figure 2**: A two-dimensional general example of when boundaries between clusters are unclear

*Figure taken from Bauckhage, 2015b*

**Results**

Results from applying the FCM variant of the soft clustering algorithm (FCM was the easiest to implement in R) to the NYC Uber data are presented in the maps below along with a map of the actual NYC Boroughs for comparison. Each cluster has been plotted in the form of a spatial heatmap overlaid on a map of New York City. Each map has been divided into a grid, where each square of the grid is colored based on the maximum probability of belonging to the given cluster within the square, with red indicating a higher probability and green indicating a lower probability.

Clusters 1, 3 and 4 are primarily in Manhattan, although cluster 4 does seem to be bleeding over into the Bronx and Queens a bit. Cluster 2 appears to be mostly a specific area in Brooklyn. And finally, cluster 5 seems that it is mostly JFK airport and some surrounding areas in queens. Additionally, notice that Staten Island is completely unrepresented, and the Bronx is essentially unrepresented as well. Furthermore, note the fuzzy “boundaries” between clusters, ie. how the red at the center of each cluster fades out gradually to green as we move from one cluster center to the next.



**Figure 3**: Results plotted on maps of NYC with borough map for comparison

Borough Map taken from: https://nycmap360.com/nyc-boroughs-map#.XEJT5FxKg2w

**Discussion**

Overall, the results of my cluster analysis are somewhat disappointing. The computed clusters do not seem to match up very well with the actual boroughs of New York City. It seems likely that this was caused by the fact that a disproportionate number of the Uber pickups were in Manhattan, causing these observations to dominate those in other boroughs. It also appears that these issues were exacerbated by cutting down the dataset to only 30,000 observations as was necessary for computational feasibility. The areas of relatively sparse data in the original dataset of over 4.5 million observations (such as Staten Island) seem to be almost non-existent after reducing the dataset down to 30,000 observations. Finding a way to run this algorithm with a bigger data sample may help to alleviate these issues, but given that soft k-means is even more computationally intensive than hard (classical) k-means, this may not be feasible.

Additionally, when the only variables present in the data are latitude and longitude of Uber pickups in 2014, it doesn’t necessarily make sense to think that the groupings identified by the clustering algorithm would match up that well with “boroughs” which were delineated in 1898 anyway. It would furthermore be interesting to see how well the clusters match the boroughs if the time information added in. This would seem to make the most sense with only the hour information, which could then be dummy coded into categories such as early morning (12am-5am), morning rush “hour” (5am-9am), midday (9am-3pm), evening rush “hour” (3pm-7pm), and late evening (7pm – 12am).

Despite the poor match with the actual NYC boroughs, much of the main point here was to visualize the “soft” nature of the clusters resulting from the soft k-means algorithm, which, again, can be seen relatively clearly in the way that the dark red fades to lighter red to green moving out from the cluster centers. The continuum style treatment of boroughs where one slowly fades into another seems to make more sense as a way of dealing with these bordering areas, rather than strict cutoffs delineating one borough vs. another.

On the whole, it seems that the soft/fuzzy version of k-means clustering has some clear advantages over the classical/hard version. The soft treatment of cluster “boundaries” seems an altogether more versatile approach, and as well seems much more realistic. Unfortunately, though it does have some weaknesses. Given the more complicated nature of computing cluster membership “indicators” (probabilities) and updating centroids, as compared with classical k-means, this method is significantly more computationally intensive. As a result of this fact, the R clustering function I used could not handle more than about 30,000 observations, which is a far cry from the over 4.5 million observations handled by the classical k-means algorithm used in the DataCamp example. Additionally, due to the far less ubiquitous nature of soft k-means (vs. hard k-means) there is a lot less support/information about this method and options for implanting it. Indeed, finding an out-of-the-box R function (or python) for executing soft k-means proved far less easy than with the classical algorithm.

These tradeoffs therefore have to be weighed in context when deciding whether the hard or soft k-means algorithm is best for a particular application. If the dataset in question lends itself to crisp, clearly delineated clusters, or is particularly large, or if the implementing analyst requires a lot of guidance/support, the classical algorithm is probably a better choice. However, if the dataset is not too large, and there is any question about to which cluster some of the datapoints belong, ie. the boundaries between clusters are not clear, the soft k-means algorithm is likely the best bet.

**References**

Bauckhage, Christian. (2015a). *Lecture Notes on Data Science: k-Means Clustering*. 10.13140/RG.2.1.2829.4886.

Bauckhage, Christian. (2015b). *Lecture Notes on Data Science: Soft k-Means Clustering*. 10.13140/RG.2.1.3582.6643.

Jaiswal, Sejal. (2018). *K-Means Clustering in R Tutorial*. DataCamp. <https://www.datacamp.com/community/tutorials/k-means-clustering-r>

**Other resources I found useful for understanding**

<https://en.wikipedia.org/wiki/K-means_clustering>

<https://en.wikipedia.org/wiki/Fuzzy_clustering>

<https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/cmeans.html>

<https://en.wikipedia.org/wiki/Softmax_function>

**Appendix**

**Link to dataset**: <https://www.kaggle.com/fivethirtyeight/uber-pickups-in-new-york-city/data>

**R code:**

# data comes as separate csv files for each month of data

# read in multiple csv files, combine into one and write to new csv

rm(list=ls())

library(dplyr)

setwd('C:\\Users\\Evan Generoli\\Downloads\\uber-pickups-in-new-york-city')

apr14 <- read.csv('uber-raw-data-apr14.csv')

may14 <- read.csv('uber-raw-data-may14.csv')

jun14 <- read.csv('uber-raw-data-jun14.csv')

jul14 <- read.csv('uber-raw-data-jul14.csv')

aug14 <- read.csv('uber-raw-data-aug14.csv')

sep14 <- read.csv('uber-raw-data-sep14.csv')

uber\_data\_14 <- bind\_rows(apr14, may14, jun14, jul14, aug14, sep14)

setwd('C:\\Users\\Evan Generoli\\Documents\\Graduate School\\Fall Semester 2018\\Statistical and Machine Learning')

write.csv(uber\_data\_14, file = 'uber\_data\_14\_combined.csv', row.names=FALSE)

# clear environment and set working directory

rm(list=ls())

setwd('C:\\Users\\Evan Generoli\\Documents\\Graduate School\\Fall Semester 2018\\Statistical and Machine Learning')

# read in & truncate dataset by taking random sample and selecting only latitude and longitude variables

# dataset must be cut down, 30k sample size is about the limit of what the clustering function could handle

library(dplyr)

set.seed(0)

df <- read.csv('uber\_data\_14\_combined.csv') %>% sample\_n(size=30000) %>% select(Lat, Lon)

summary(df)

head(df)

# implement clustering algorithm with 5 clusters, don't standardize variables

# metric must be specified as squared euclidean for this be soft k-means/fuzzy C-means

library(cluster)

cluster <- fanny(x=df, k=5, metric='SqEuclidean', stand=F)

membership\_probs <- 100 \* as.data.frame(cluster$membership)

colnames(membership\_probs) <- c('Cluster\_1','Cluster\_2','Cluster\_3','Cluster\_4','Cluster\_5')

head(membership\_probs)

####################################### create maps of results

# in order to download map from google,

# an api key must be obtained online through a billing enabled google account

# then api key must be registered

# create dataframe for maps

df\_map <- bind\_cols(membership\_probs, df) %>% na.omit()

df\_map %>% head()

# install developer version of ggmap in order to register api key with register\_google() function

library(devtools)

devtools::install\_github("dkahle/ggmap", ref = "tidyup")

library(ggmap)

register\_google(key = 'AIzaSyA-GSv-qhnapRyYXkr-x64qJGg3lMRu0V8')

# get map of nyc

nyc\_map <- get\_map("New York", zoom = 10)

ggmap(nyc\_map)

# overlay spatial heatmap of cluster probabilities on nyc map for each cluster

ggmap(nyc\_map, extent = "device") +

stat\_summary\_2d(data = df\_map, aes(x = Lon, y = Lat, z = Cluster\_1),

fun = max , alpha = 0.6, bins = 250) +

scale\_fill\_gradient(name = "% Probability", low = "green", high = "red") +

ggtitle('Cluster 1')

ggmap(nyc\_map, extent = "device") +

stat\_summary\_2d(data = df\_map, aes(x = Lon, y = Lat, z = Cluster\_2),

fun = max , alpha = 0.6, bins = 250) +

scale\_fill\_gradient(name = "% Probability", low = "green", high = "red") +

ggtitle('Cluster 2')

ggmap(nyc\_map, extent = "device") +

stat\_summary\_2d(data = df\_map, aes(x = Lon, y = Lat, z = Cluster\_3),

fun = max , alpha = 0.6, bins = 250) +

scale\_fill\_gradient(name = "% Probability", low = "green", high = "red") +

ggtitle('Cluster 3')

ggmap(nyc\_map, extent = "device") +

stat\_summary\_2d(data = df\_map, aes(x = Lon, y = Lat, z = Cluster\_4),

fun = max , alpha = 0.6, bins = 250) +

scale\_fill\_gradient(name = "% Probability", low = "green", high = "red") +

ggtitle('Cluster 4')

ggmap(nyc\_map, extent = "device") +

stat\_summary\_2d(data = df\_map, aes(x = Lon, y = Lat, z = Cluster\_5),

fun = max , alpha = 0.6, bins = 250) +

scale\_fill\_gradient(name = "% Probability", low = "green", high = "red") +

ggtitle('Cluster 5')