

OPTIMAL POSITIONING OF REFUELLING STATION

TABLE OF CONTENTS

1. INTRO	03
1.1 Introduction	03
2. DATA EXPLORATION	04
2.1 DATA OVERVIEW	04
2.2 DATA CLEANING OVERVIEW	05
3. MODEL BUILDING STEPS	08
3.1 CLUSTER VALIDATION	09
3.2 CLUSTER MAKING	13
4. CONCLUSION AND FUTURE WORK	17
4.1 CONCLUSION	17
4.2 REFERENCE	17

CHAPTER 1

INTRODUCTION

Optimal Positioning of Refueling Station in Portugal

Positioning of Fueling Station is very critical for any city. Setup cost of a Fuel station is very high, because of Ground exploration, Land examination, locality, judgement of quantity of traffic, etc. This analysis can be used to determine the positioning of Fuel Station setups.

Process of Analysis

The steps involved in the study are

- Getting Overview of Data
- Preprocessing of data & Data Cleaning
- Making a smaller sample of the data from the original training set. The training set was of 2gb. As per hardware availability, a smaller sample of 500MB data was made.
- Get distance deviation of start and end points of any journey from closest Taxi Stands
- As a cab running distance length before refueling, had to be assumed, 50Km was assumed as the distance. So, A fuel pump should be between 50 km. For that 45 km length was presumed.
- Long journeys (greater than 50 km) were identified from the Polyline co-ordinates.
- The places which are in 45 km interval (apparently), were identified. From those places, duplicate entries and the number of occurrences were saved separately to get the importance of traffic rush.
- Clustering (k-Medoids implementation of R -> PAM) was used. Comparison of numbers of clusters were done
- Cluster was made using Taxi Stands and Outskirt locations of Porto.

CHAPTER 2

DATA EXPLORATION

DATA OVERVIEW

Data is available form

1. <http://archive.ics.uci.edu/ml/datasets/Taxi+Service+Trajectory+-+Prediction+Challenge%2C+ECML+PKDD+2015>
2. <https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/data>

Initially data looked like following:-

	TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID	TIMESTAMP	DAY_TYPE	MISSING_DATA	POLYLINE
1	1.372637e+18	C	NA	NA	20000589	1372636858	A	False	[[[-8.618643,41.141412],[[-8.618499,41.141376],[[-8.62...
2	1.372637e+18	B	NA	7	20000596	1372637303	A	False	[[[-8.639847,41.159826],[[-8.640351,41.159871],[[-8.64...
3	1.372637e+18	C	NA	NA	20000320	1372636951	A	False	[[[-8.612964,41.140359],[[-8.613378,41.14035],[[-8.614...
4	1.372637e+18	C	NA	NA	20000520	1372636854	A	False	[[[-8.574678,41.151951],[[-8.574705,41.151942],[[-8.57...
5	1.372637e+18	C	NA	NA	20000337	1372637091	A	False	[[[-8.645994,41.18049],[[-8.645949,41.180517],[[-8.646...
6	1.372637e+18	C	NA	NA	20000231	1372636965	A	False	[[[-8.615502,41.140674],[[-8.614854,41.140926],[[-8.61...

This data consists of 9 variables and 1710670 rows. Each row corresponds to a trip metadata.

The columns contain the following variables:

1. **TRIP_ID:** (String) It contains an unique identifier for each trip;
2. **CALL_TYPE:** (char) It identifies the way used to demand this service. It may contain one of three possible values:
 1. 'A' if this trip was dispatched from the central;
 2. 'B' if this trip was demanded directly to a taxi driver on a specific stand;
 3. 'C' otherwise (i.e. a trip demanded on a random street).
3. **ORIGIN_CALL:** (integer) It contains an unique identifier for each phone number which was used to demand, at least, one service. It identifies the trip's customer if CALL_TYPE='A'. Otherwise, it assumes a NULL value;
4. **ORIGIN_STAND:** (integer): It contains an unique identifier for the taxi stand. It identifies the starting point of the trip if CALL_TYPE='B'. Otherwise, it assumes a NULL value;

5. **TAXI_ID**: (integer): It contains an unique identifier for the taxi driver that performed each trip;
6. **TIMESTAMP**: (integer) Unix Timestamp (in seconds). It identifies the trip's start;
7. **DAYTYPE**: (char) It identifies the daytype of the trip's start. It assumes one of three possible values:
 1. 'B' if this trip started on a holiday or any other special day (i.e. extending holidays, floating holidays, etc.);
 2. 'C' if the trip started on a day before a type-B day;
 3. 'A' otherwise (i.e. a normal day, workday or weekend).
8. **MISSING_DATA**: (Boolean) It is FALSE when the GPS data stream is complete and TRUE whenever one (or more) locations are missing
9. **POLYLINE**: (String): It contains a list of GPS coordinates (i.e. WGS84 format) mapped as a string. The beginning and the end of the string are identified with brackets (i.e. [and], respectively). Each pair of coordinates is also identified by the same brackets as [LONGITUDE, LATITUDE]. This list contains one pair of coordinates for each 15 seconds of trip. The last list item corresponds to the trip's destination while the first one represents its start;

DATA CLEANING & SMALLER SAMPLE PRODUCING

The entries where Polyline Data was missing, that means Missing_Data field was true, were removed. Because if the Polyline information was missing, the entry is not suitable for further analysis.

Small Sample producing:-

As CALL_TYPE information will be used later, subset should be made keeping the same ratio of CALL_TYPE factors. Initially it was:-

<pre>> table(data1\$CALL_TYPE)</pre>	<pre>> prop.table(table(data1\$CALL_TYPE))</pre>
<div style="display: flex; justify-content: space-around;"> A B C </div> <div style="display: flex; justify-content: space-around;"> 364770 817881 528019 </div>	<div style="display: flex; justify-content: space-around;"> A B C </div> <div style="display: flex; justify-content: space-around;"> 0.2132322 0.4781057 0.3086621 </div>

Main training dataset is of 2gb. As there was hardware barrier, I had to make a smaller sample keeping above ratio intact.

```

##attaching a Row Id Column for subsetting index
data1$rowID <- c(1:1710670)
#Take Sample from each call type(1:4 Ratio)
b <- sample(data1$rowID[data1$CALL_TYPE=='A'],91192)
c <- sample(data1$rowID[data1$CALL_TYPE=='B'],204470)
d <- sample(data1$rowID[data1$CALL_TYPE=='C'],132005)

#Subsetting with selected RowID
data3 <- subset(data1,rowID %in% b)
data4 <- subset(data1,rowID %in% c)
data5 <- subset(data1,rowID %in% d)

#merging 3 subsets
dataSmall <- rbind.data.frame(data3,data4)
dataSmall <- rbind.data.frame(dataSmall,data5)

```

A dataFrame named SmallData were made by keeping the ratios intact. Its size was 500MB. ¼ th of original dataset.

Location data of nearby stands and landmarks were kept separately in “locData” data frame.

Five new fields were added in the SmallData- Start, End, DIST, StartDev, EndDev.

Start- Closest Taxi Stand/Pre given Land Mark of starting point of the trip.

End- Closest Taxi Stand/Pre given Land Mark of ending point of the trip.

DIST- Distance of Starting point and Ending of trip.

StartDev- This is the deviation of closest land mark and the starting point of the trip.

EndDev- This is the deviation of closest land mark and the ending point of the trip.

For distance calculation **Earth curvature distance method** used. Google Api for driving distance calculation would be the perfect one. But As it supports 2500 request max per day, I had to go for alternatives. A method from “**IMAP**” package, named **gdist()** or following method can be used for that. Both are same apparently.

```

earth.dist <- function (long1, lat1, long2, lat2)
{
  rad <- pi/180
  a1 <- lat1 * rad
  a2 <- long1 * rad
  b1 <- lat2 * rad
  b2 <- long2 * rad
  dlon <- b2 - a2
  dlat <- b1 - a1
  a <- (sin(dlat/2))^2 + cos(a1) * cos(b1) * (sin(dlon/2))^2
  c <- 2 * atan2(sqrt(a), sqrt(1 - a))
  R <- 6378.145 #Radius of Earth
  d <- R * c
  return(d)
}

```

After those steps, dataframe looked like following:-

	TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID	TIMESTAMP	DAY_TYPE	MISSING_DATA	POLYLINE
1	1.399246e+18	C	NA	NA	20000233	1399246019	A	False	[[-8.628021,41.158242],[8.627634,41.158557],[8.62...
2	1.394666e+18	A	2002	NA	20000325	1394665692	A	False	[[-8.620101,41.146839],[8.620092,41.146848],[8.62...
3	1.385978e+18	C	NA	NA	20000473	1385977724	A	False	[[-8.571231,41.159457],[8.569611,41.161077],[8.56...
4	1.377607e+18	B	NA	33	20000013	1377607046	A	False	[[-8.600148,41.182704],[8.600256,41.182713],[8.59...
5	1.375806e+18	B	NA	28	20000619	1375806206	A	False	[[-8.584344,41.163102],[8.58492,41.162913],[8.586...
6	1.373310e+18	B	NA	15	20000367	1373309781	A	False	[[-8.585658,41.148558],[8.585712,41.148918],[8.58...

Start	End	DIST	StartDev	EndDev
H. Militar	Santa Maria	2.677448803	0.104245443	1.577273704
Carregal	Santa Maria	4.748250021	0.169792697	5.715195006
São Roque	Santa Maria	1.608408146	0.730468787	1.828875121
H. São João	Santa Maria	3.775374882	0.009486562	6.368154374
Dragão	Santa Maria	2.172511881	0.028304757	3.798084268
Campanhã	Santa Maria	1.665632144	0.019908297	0.719535249

CHAPTER 3

MODEL BUILDING STEPS

Main Steps are following:-

1. **Identify those trips where refueling may be needed-** Here three conditions were checked.
 - The trips where (Call type is 'C' AND ((Start point deviation is greater than 50km and trip length is greater than 50km) OR Endpoint Deviation is greater than 50km) AND Start Point Deviation is less than 250Km AND end point deviation is less than 250Km) ..Above 250 Km deviated points are discarded because they were incomplete data and Google API could not find any Location, as they were pointing to North Atlantic Ocean.
 - The trips where (Call type is 'A' AND trip length was greater than 50km)
 - The trips where (Call type is 'B' AND trip length was greater than 50km)
2. After this, points from apparently 45Km intervals were selected and Place name was retrieved by **Google Reverse Geocode** API and stored into a data frame with **LatLon**.
3. **Google Reverse Geocode** API gave multiple with slightly different LatLon same place name. In this case, count of those repetition were stored as High Traffic Area in PlaceCount data frame. And mean of their LatLon were used further.
4. **Two Distance Matrix was prepared.** One for the previously mentioned Landmarks/Taxi Stands, another one is Out Skirt locations, which were identified from the trips. Distance Matrix were calculated by **gdist() method of IMAP** package. **If an organization could acquire corporate license to use Google API, then driving distance can be fetched from Google Driving Distance API too.** Method will be pointed in the script file. Upper triangular matrix was calculated and it was copied to lower triangular Matrix, to make the operation faster. Two Distance Matrix were named **distanceMatrix**, **distanceMatrixFarLoc**.
5. **K-Medoids** implementation of **cluster** package, **PAM (Partition Around Medoids)**, was found useful. As k-Means clustering does not take custom distances, an alternative was needed. I found **PAM** very helpful as it takes distance matrix also.
6. Before making the cluster, validation of number of clusters should be done to determine perfect number for cluster. Those steps are explained in following section.

CLUSTER VALIDATION

A package called **clValid** is used to do internal and stability validation. Validation measures are following.

Internal measures-

For internal validation, we selected measures that reflect the compactness, connectedness, and separation of the cluster partitions.

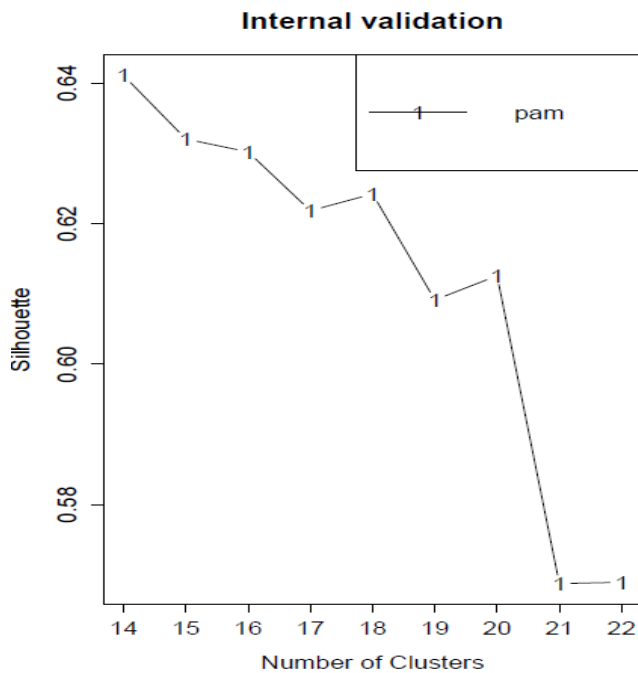
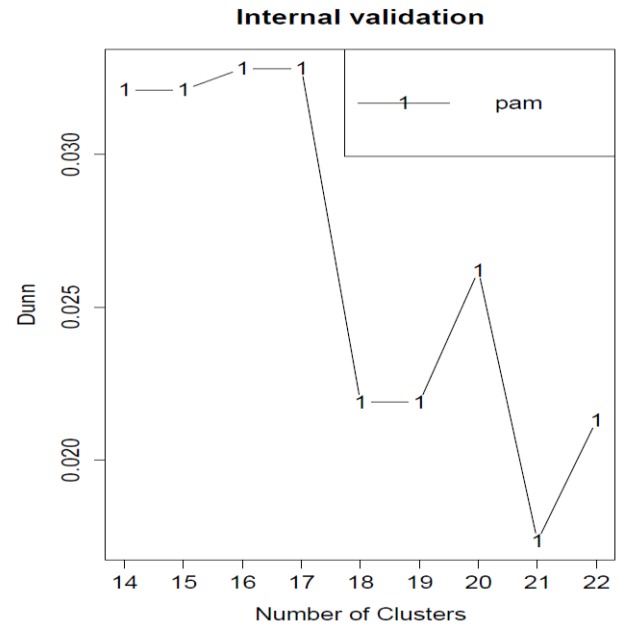
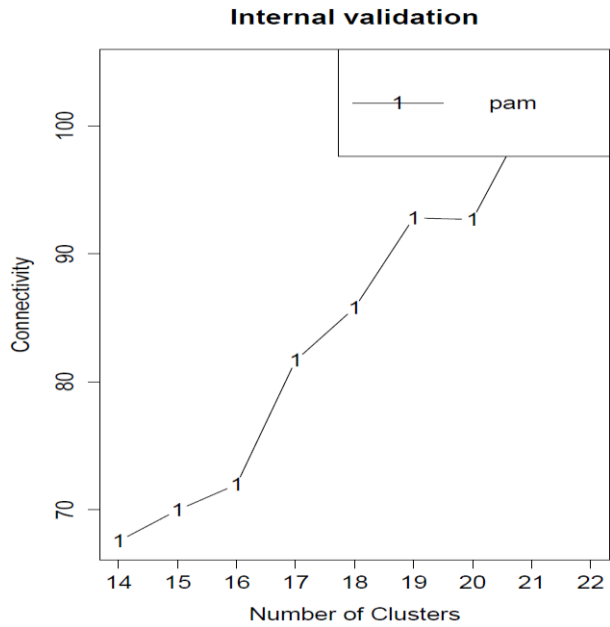
- Connectivity- The connectivity can be between 0 and Infinity, and should be **minimized**.
- Dunn Index- The Dunn index aims to identify dense and well-separated clusters. It is defined as the ratio between the minimal inter-cluster distance to maximal intra-cluster distance. It should be **maximized**
- Silhouette Co-efficient- The silhouette coefficient contrasts the average distance to elements in the same cluster with the average distance to elements in other clusters. Objects with a high silhouette value are considered well clustered, objects with a low value may be outliers. It should be **maximized**.

Stability measures- Average Proportion of Non-overlap (APN), Average Distance (AD), Average Distance between Means (ADM) and Figure of Merit (FOM) were checked. All should be **minimized**.

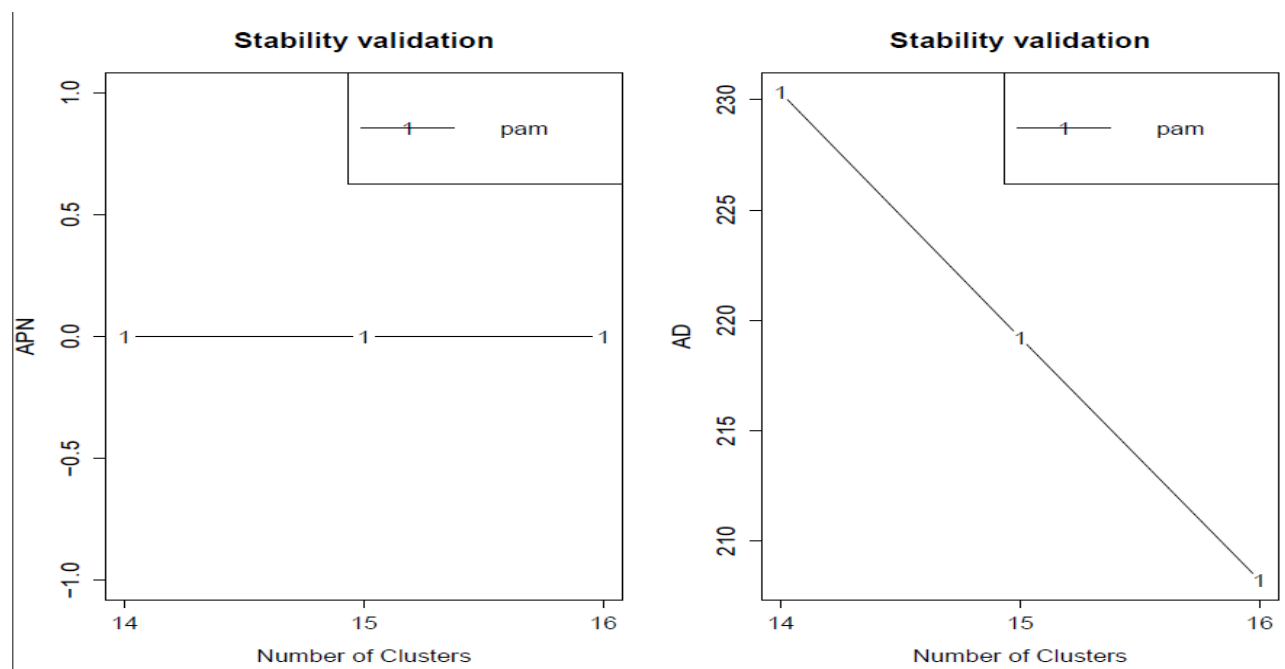
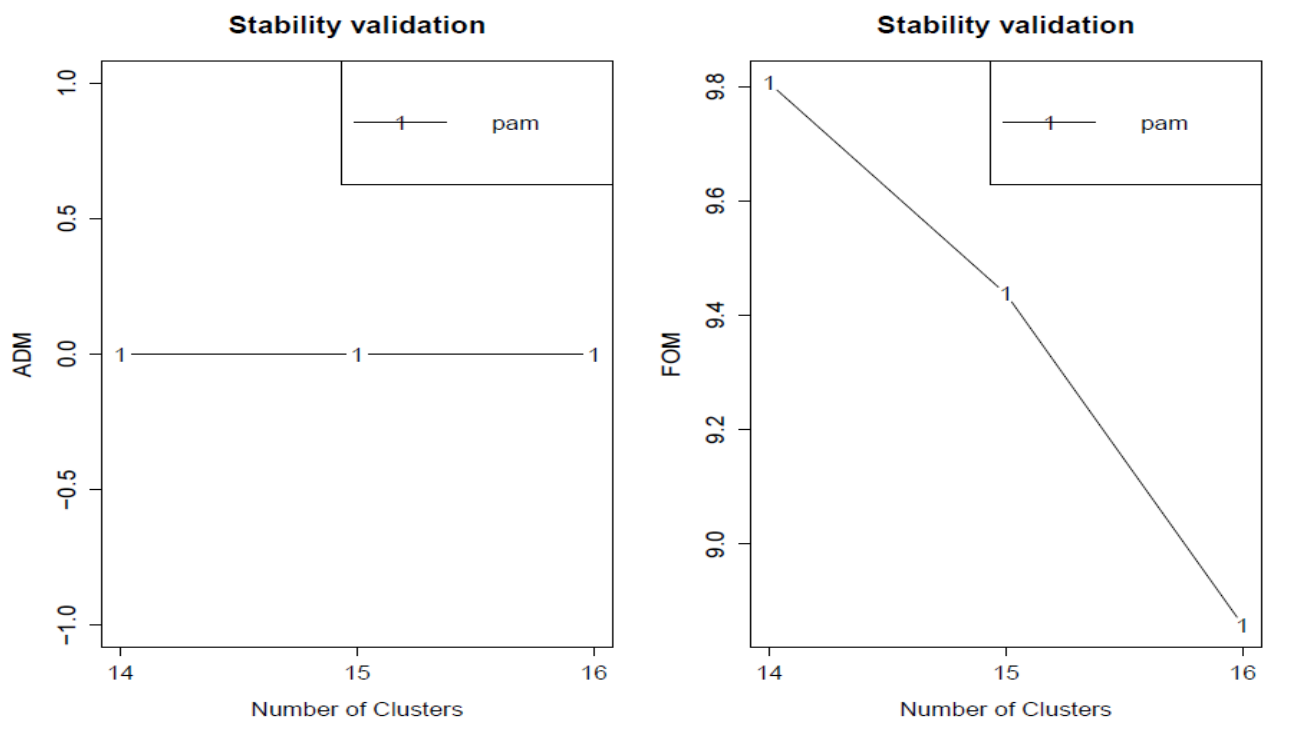
Primarily as rule of thumbs from $\sqrt{N/2}$ to \sqrt{N} number of clusters were chosen for validation.

After generating the result from internal validation, possible three possible number of clusters were checked for stability validation. Diagrams are following.

OutSkirt Points' Cluster validation diagrams:-

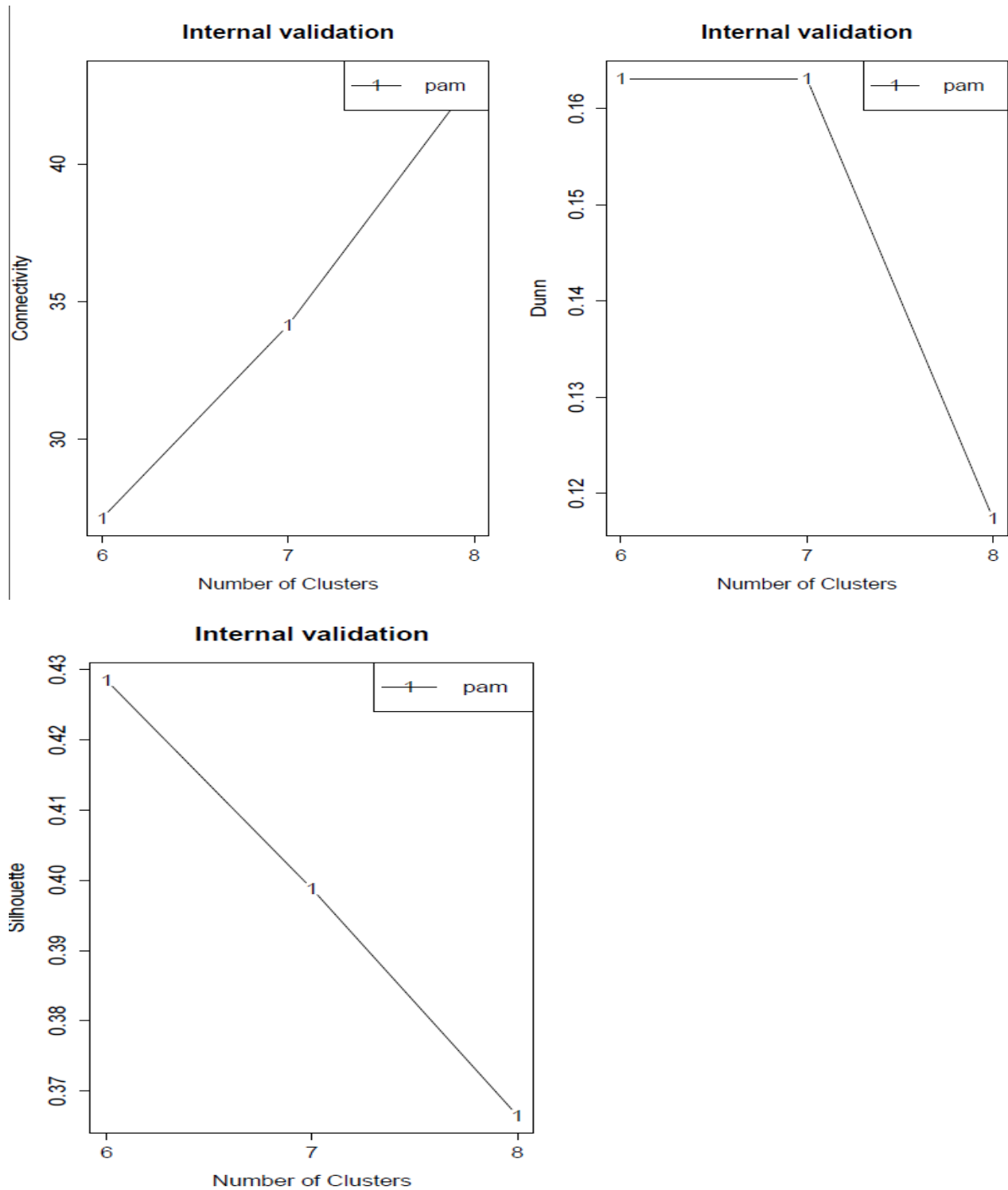


Based on this diagram possible number of clusters were taken as **(14,15,16)** . With those Stability validation were checked.



As we can see from the above diagram, **perfect number of cluster will be 16.**

Now, Taxi Stand/Given Land Marks' Cluster Validation diagrams:-



As we can see from the above diagram, **perfect number of cluster will be 6 for Taxi Stands/Given Land Marks.**

CLUSTER MAKING

Two clusters were made by Partition around Medoids. ClusterOutSkirts and ClusterStands.

```
ClusterOutskirts <- pam(distanceMatrixFarLoc,16)
```

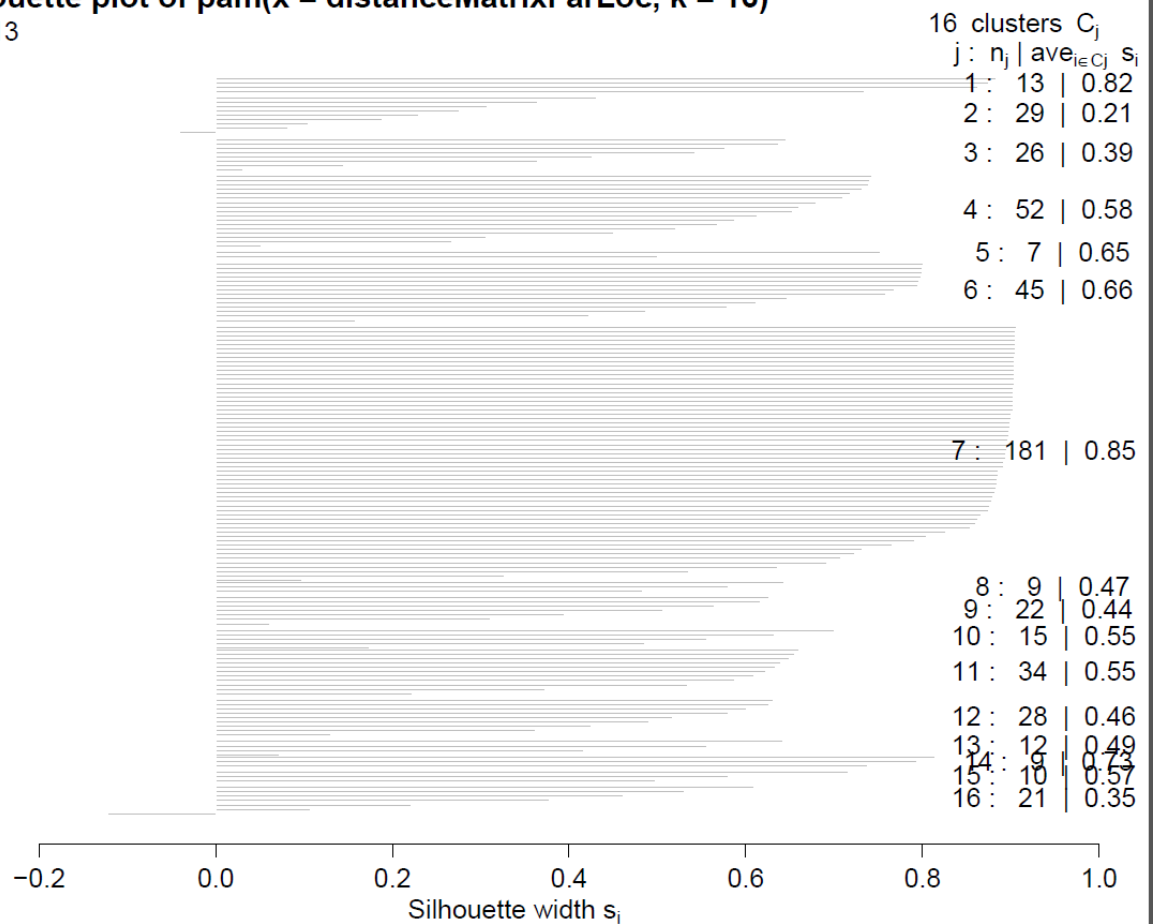
```
clusterStands <- pam(distanceMatrix,6)
```

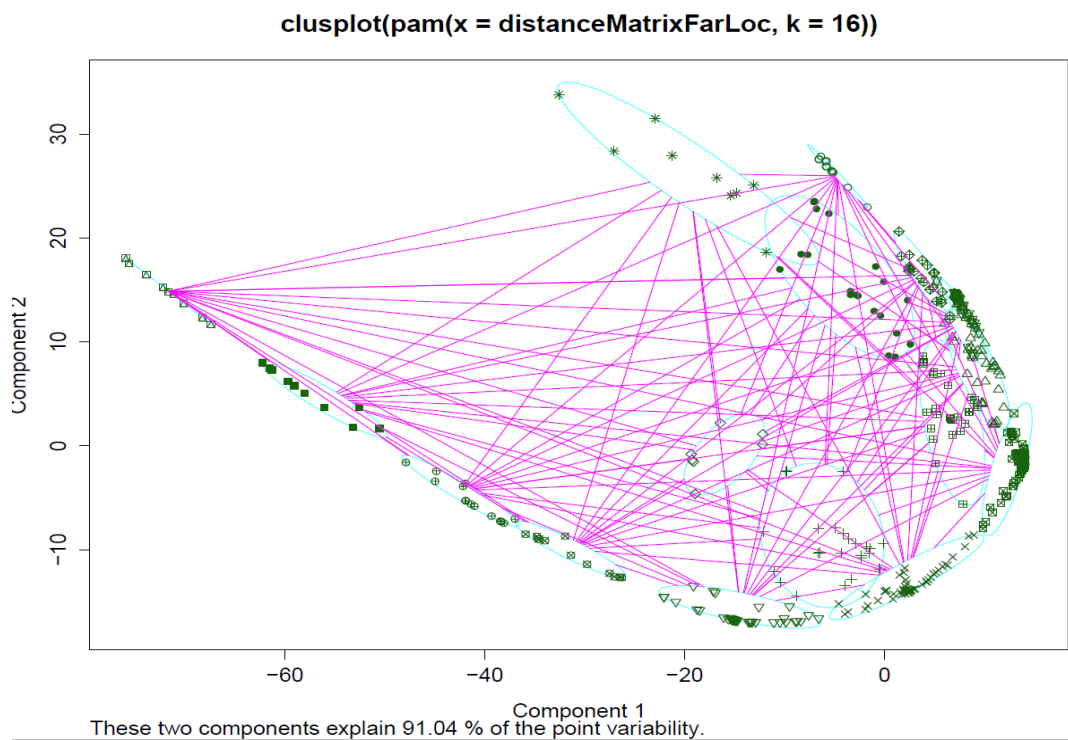
Cluster plots are following:-

OutSkirt Cluster:-

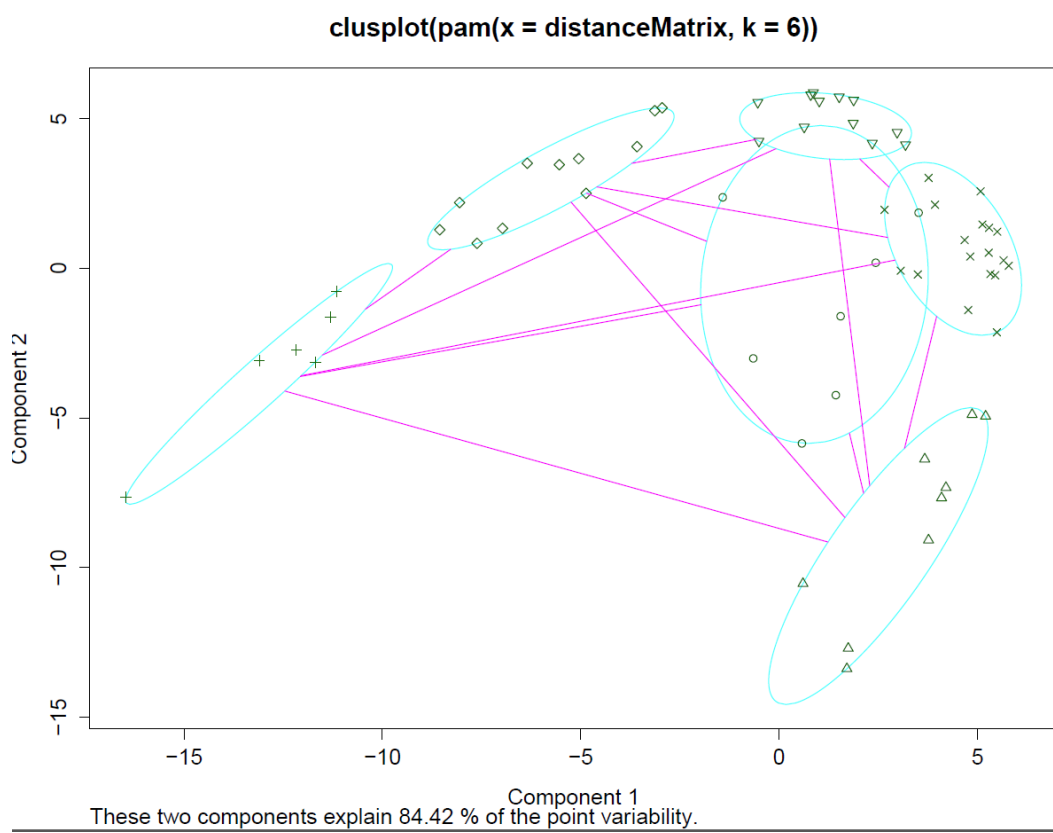
Silhouette plot of pam(x = distanceMatrixFarLoc, k = 16)

n = 513



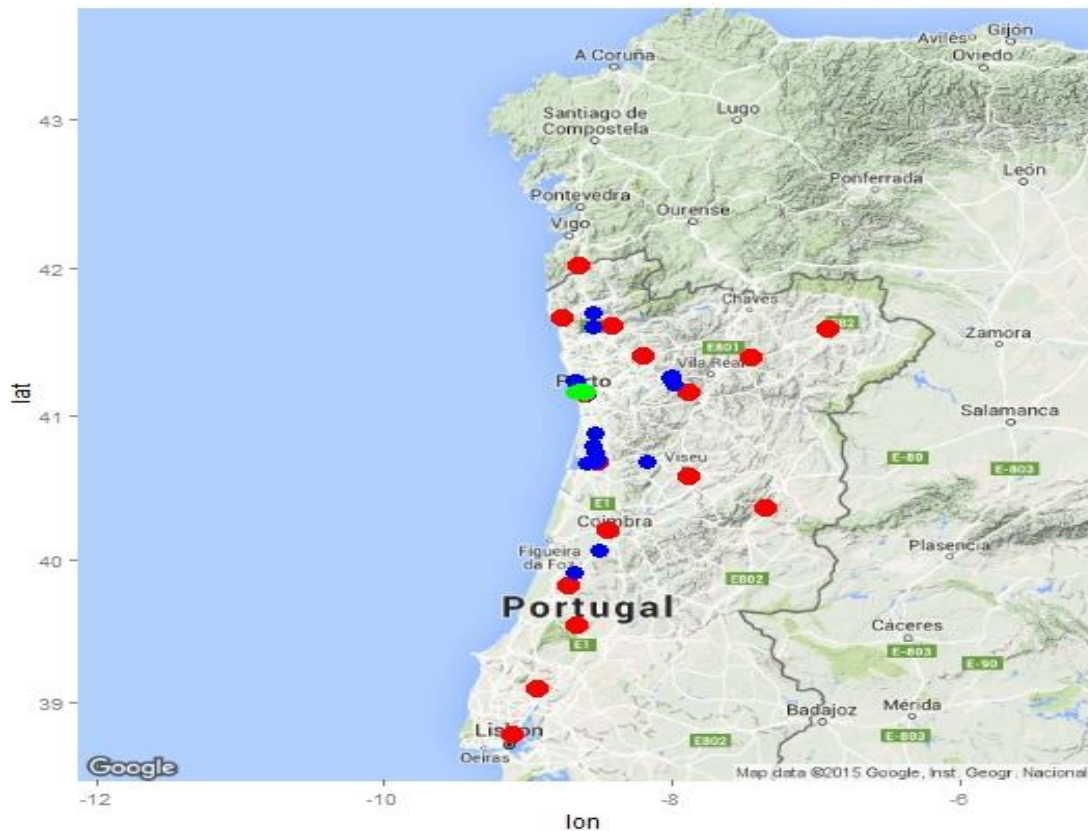


TaxiStandCluster



RESULTS

After plotting the points in the map following image was generated:-





















Red Points are the OutSkirt Cluster medoids, Blue points are the high traffic areas, and the green points are Fuel Pumps at the City of Porto.

This data can be submitted to higher authority to decide the perfect locations with all conditions met using the derived possible places.

This image was generated from one fourth of the dataset. Still it is showing apparently correct locations. As from Porto to Lisbon is almost 313km by drive, it has 5 Red dots in between almost following the A1 Road.

Object description of workspace image:-

Data		
▶ BetterChoiceOfFu...	21 obs. of 5 variables	
▶ ClusterMembers	513 obs. of 2 variables	
▶ dataSmall	427663 obs. of 15 variables	
distanceMatrix	num [1:63, 1:63] 0 2.803 4.773 3.845 0.752 ...	
▶ distanceMatrixFa...	Large matrix (263169 elements, 2.1 Mb)	
▶ FiftykmDist	257 obs. of 15 variables	
▶ FiftykmDist_A	9 obs. of 15 variables	
▶ FiftykmDist_B	66 obs. of 15 variables	
▶ FuelLocPrimary	16 obs. of 4 variables	
▶ FuelLocStation	6 obs. of 4 variables	
▶ locData	63 obs. of 4 variables	
▶ PlaceCount	21 obs. of 2 variables	
▶ SecondLocData	773 obs. of 3 variables	
▶ SecondLocDataTri...	513 obs. of 3 variables	
Values		
▶ ClusterOutskirts	Large pam (10 elements, 2.3 Mb)	
▶ clusterStands	List of 10	
▶ internalValidator	Large clValid (996.2 Kb)	
▶ stabilityValidat...	Formal class clValid	
▶ StandInternalVal...	Formal class clValid	
▶ StandStabilityVa...	Formal class clValid	
▶ testmap	Large ggmap (409600 elements, 3.1 Mb)	
Functions		
earth.dist	function (long1, lat1, long2, lat2)	
GeoDistanceInMet...	function (df.geopoints)	
latlon2ft	function (origin, destination)	
reverseGeoCode	function (latlng)	

- **BetterChoiceOfFuelPumps**- Some high traffic areas with LatLon
- **ClusterMembers**- 513 areas with their cluster ids
- **dataSmall**- Small dataset of 500mb

- **distanceMatrix**- Distance Matrixes of Given Taxi Stands/LandMarks and derived locations from trips
- **FiftykmDist***- DataSets which were filtered for long trips with different call types
- **FuelLocPrimary**- 16 cluster medoids
- **FuelLocStation**- 6 Medoids of pre-given stands
- **Place Count**- Some high traffic areas . Same as **BetterChoiceOfFuelPumps**, without LatLon
- **SecondLocData**- Identified locations from the trips.
- **SecondLocDataTrimmed**- SecondLocData after removing repetition and assigning mean of the LatLong to corresponding place.
- **reverseGeoCode**- Function to call google api to get a place name.
- **latlon2ft & earth.dist**- Method to get the distance between two points. Latlon2ft uses Google Driving distance API, but it could not be used due to per day limit of API call.

CHAPTER 4

CONCLUSION AND FUTURE WORK

- It can be perfected using driving distance API, if corporate license can be acquired for GoogleAPI usage
- Parallel implementation of this process using MapReduce to make this process work where humongous data will come.

REFERENCES

- <http://www.researchgate.net/publication/259172799> A computer model for optimizing the location of natural gas fueling stations
- <http://www.ajer.org/papers/v3%289%29/T03901470158.pdf>
- <https://en.wikipedia.org/wiki/K-medoids>
- <https://cran.r-project.org/web/packages/clValid/vignettes/clValid.pdf>
- Etc.