CSC 791 - IOT Analytics

Bandwidth allocation in femtocells based on movement patterns of pedestrians

December 4, 2016

Project Report

BY

Neetish Pathak

Ronak Doshi

Vivek Subbarao

Table of Contents

Abstract

As discussed in the introductory lecture for the course CSC 791, IoT links smart objects to the Internet, and enables an exchange of data never available before. It is estimated that IoT will consist of 50 billion devices connected to the Internet by 2020. With more and more devices around us turning smarter day by day, there is a major challenge for making the optimum usage of available resources on these devices to ensure long term benefits. We are in the era where most of the connectivity happens on wireless and as stated in the paper - Femtocell Networks: A Survey [1], the surest way to increase the system capacity of a wireless link is by getting the transmitter and receiver closer to each other, which creates the dual benefits of higher quality links and more spatial reuse. One of the devices that can facilitate this stronger connectivity between a nomadic cellphone user and the network is a femtocell. Femtocell is a data access point installed by home users to get better indoor voice and data coverage. Now, keeping the mobile nature of users in perspective, we need to device a mechanism the optimally allocate bandwidth on these femtocells. In this report, we discuss the a time-forecasting based model to decide the bandwidth (or other resource) allocation on the femtocells. It is quite obvious that the bandwidth allocation will be directly proportional to the demand and demand can be measured by the density of users around the femtocell. So, our objective is to find the density of pedestrians around the femtocells that can facilitate optimal resource allocation.

Problem Under Study

Original Problem Statement

Allocate b/w on femtocells by predicting the path a user follows. IoT data for paths people follow in Copenhagen and other cities are available

The original problem asked for predicting pedestrians’ path in a city that would facilitate b/w allocation on the femtocells. After analysis of the problem statement, we decided that the data we need in order to achieve a solution to the above problem is as follows:

1. Personal trajectory data in (x,y) and timestamps of the pedestrians in a virtual city
2. Timestamp of the pedestrians when moving near a femto-cell

However, an alternative way of looking at the problem is to predict the density of pedestrians around a certain femtocell in future so that the b/w and power resources on the femtocell can be utilized accordingly.

With the same requirement as the above problem statement, this problem looked straightforward and more closer to the solution. So we modified the problem slightly.

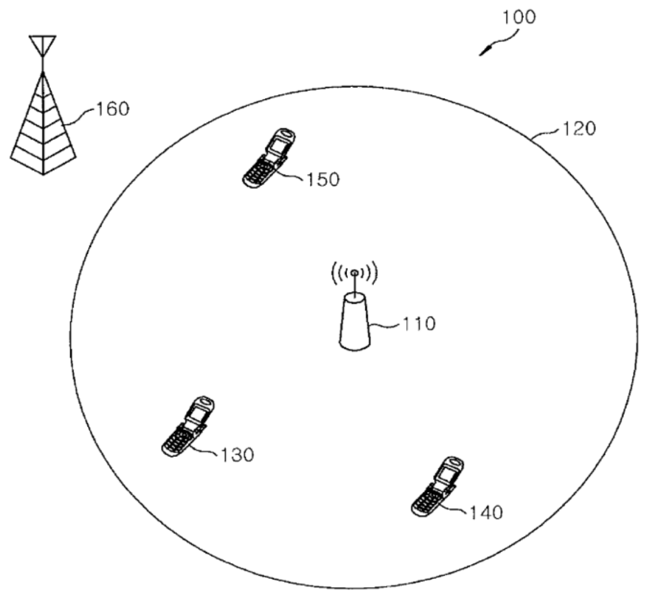
Modified Problem Statement:

Predict the density (paths) of pedestrians near femtocells in a city to facilitate adaptive bandwidth allocation in future

It would be helpful if we get a little background of the femtocells and its use.

What is a femtocell?

Femtocells are short range, low cost and low power base-stations installed for better voice and data reception.The femtocell device communicates with the cellular network over a broadband connection like DSL or cable modem. Below diagram shows a femtocell (marked 10) device between multiple mobile phone users. The distance between the user and a femtocell is lesser as compared to distance between the user and mobile base station (marked 160). This ensures longer battery life of mobile devices.



Advantages of using femtocells:

1. Ensures cellular data systems provide service roughly comparable to that offered by Wi-Fi networks
2. Due to their short transmit-receive distance, femtocell scan greatly lower transmit power and ensure prolong handset battery life
3. There is very little upfront cost to the service provider

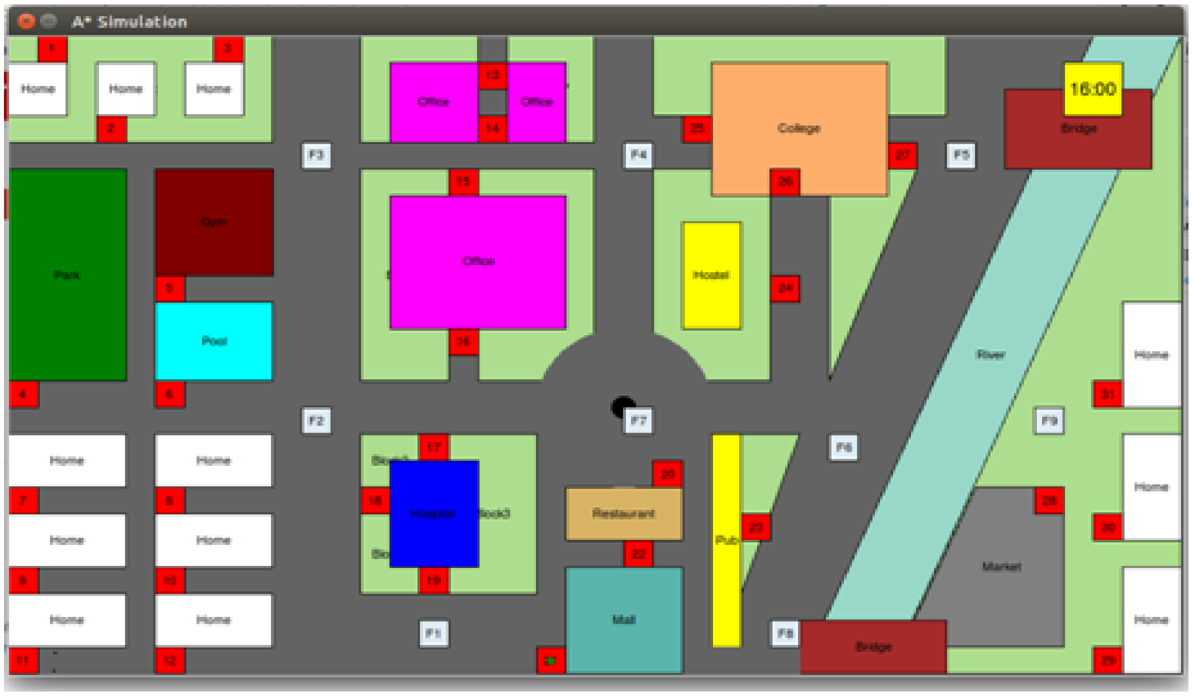
Femtocell can provide quality service to a limited no. of users and hence it makes sense to ensure optimum utilization of resources on the femtocells using the adaptive bandwidth and power control strategies in the close vicinity.

Proposed Solution

Following is the proposed solution for the bandwidth allocation problem on the femtocells. The solution as of now has the project scope and based on time-forecast model. It can be extended in future to HMM based model [2].

1. Simulate the Pedestrian Movement in a virtual city based on certain goals
2. Collect positions and timestamps of pedestrian movement
3. Collect timestamps of the users near femtocells
4. Data visualization and modelling
5. Predict the density of pedestrians around femtocells for the b/w allocation

Simulation: Virtual City



We created a virtual city using graphics.py library [3] & [4] in python. A configuration file was created to place the obstacles, origin points and destinations in the map. The red squares of unit dimension in the map represent the source and destinations for the user. The white cells of unit dimension are the femtocells placed at strategic locations in the city. As the pedestrians pass by these femtocells, the time of the day is registered. (A video of the simulation is uploaded on https://youtu.be/e28YV-466MU)

A user/pedestrian randomly gets generated in the system and a set of source and destination pair is randomly generated for him. The pedestrian moves in the virtual city and the time stamps as he goes through each of the grids is registered. We simulate the same scenario for 100 pedestrians for 24 days and collect the data as per their movement in the system. Also, as the user crosses any femtocell, the time stamp is noted which is used for calculating the density of pedestrians around the that particular femtocell in future. Our objective to register the pedestrian movement which is totally random in this case. A part of these readings are treated as training data and the rest as test data for predicting the density of users around the femtocells.

The movement of a pedestrian from the source to destination is guided by the A* shortest path algorithm which is explained below.

Path Finding Problem

Path planning consists of finding an optimal path (generally the shortest one) between a starting point and a destination point in a virtual environment, avoiding obstacles. Traditionally, path planning has been solved using a heuristic search algorithm such as A* directly coupled with the low-level animation of the pedestrian. The use of A* for path planning is based on a two step process. The virtual environment is first divided to produce a grid of cells. This grid is formally equivalent to a connectivity tree of branching factor eight, as each cell in the divided environment has eight neighbors. Searching this connectivity tree with A* using a distance-based heuristic (Euclidean distance or Manhattan distance) produces the shortest path to the destination point. This path is calculated offline, as A* is not a real-time algorithm, and the pedestrian is subsequently animated along this path.

In order to let an object or character move inside a scene from one location to another, a path has to be planned that guarantees a collision-free translation from the start to the goal position. Hence, the whole task of path planning is usually broken down into three sub-problems:

• First, one has to find a suitable division of the ground on which one can build a graph. This can be done offline in a pre-processing step. The resulting graph should be as lean as possible to allow a fast search. If the graph is too large, the search will be significantly slowed down. One the other hand, the division should be as fine as possible so that the areas corresponding to graph nodes are not too large. This would lead to an approximation error which ends up in suboptimal paths.

• Then, the graph has to be searched for a solution which connects the found nodes. For static environments, as expected, the A* algorithm is commonly used.

• Afterwards, the resulting sequence of graph nodes needs to be transferred back to the original environment.

A* algorithm:

A* is a computer algorithm that is widely used in pathfinding and graph traversal, the process of plotting an efficiently traversable path between multiple points, called nodes. It enjoys widespread use due to its performance and accuracy.

A* is an informed search algorithm, meaning that it solves problems by searching among all possible paths to the solution (goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that appear to lead most quickly to the solution. It is formulated in terms of weighted graphs: starting from a specific node of a graph, it constructs a tree of paths starting from that node, expanding paths one step at a time, until one of its paths ends at the predetermined goal node.

At each iteration of its main loop, A* needs to determine which of its partial paths to expand into one or more longer paths. It does so based on an estimate of the cost (total weight) still to go to the goal node. Specifically, A* selects the path that minimizes

$f(n) = g(n) + h(n)$

where n is the last node on the path, g(n) is the cost of the path from the start node to n, and h(n) is a heuristic that estimates the cost of the cheapest path from n to the goal. The heuristic is problem-specific. For the algorithm to find the actual shortest path, the heuristic function must be admissible, meaning that it never overestimates the actual cost to get to the nearest goal node.

During the search the A* algorithm maintains two lists of nodes: The open list contains the nodes that have to be considered next and the closed list which contains the nodes already visited. The algorithm itself consists of expanding the one node from the open list, whose fitness function is minimal. Expanding a node means putting it into the closed list and inserting the neighbors into the open list and evaluating the fitness function. The algorithm stops, when the goal node gets expanded. The choice of a good heuristic is necessary in order to achieve both quality and efficiency of the search. As long as the heuristic underestimates the real cost, the shortest path is guaranteed to be found. Nevertheless, underestimating can easily lead to an expansion of too many nodes. But when the heuristic is allowed to overestimate the remaining cost, faster results can be achieved because fewer nodes get expanded.

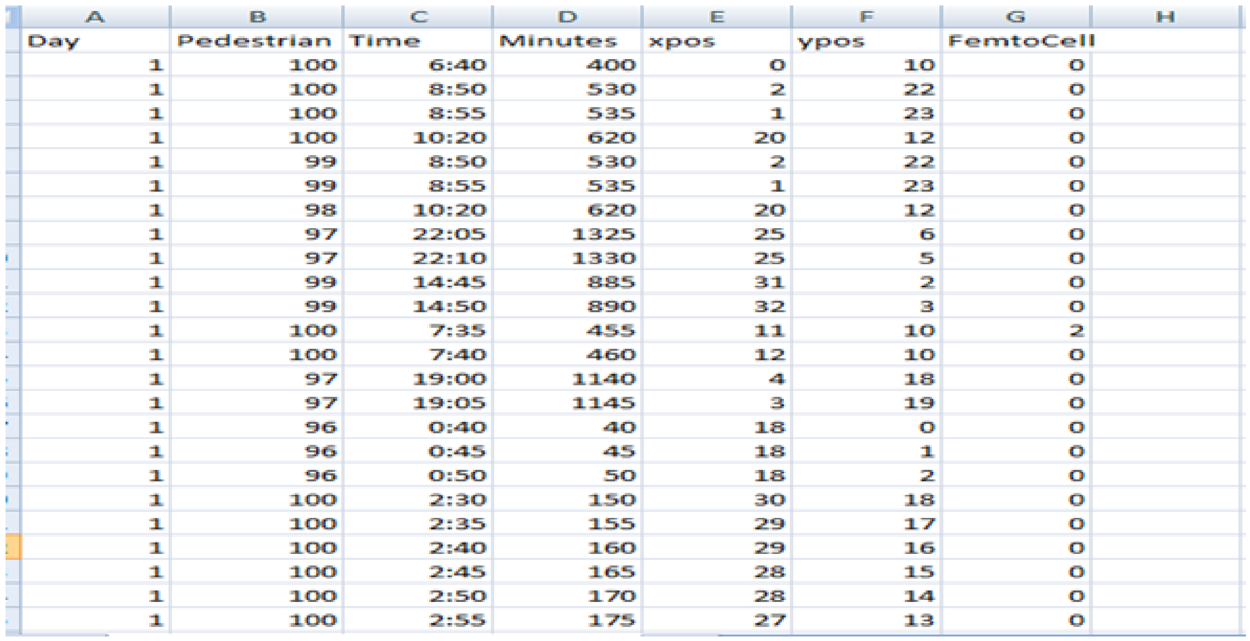
If overestimating the distance to the goal, the A* algorithm tends to expand nodes that lie on the direct path to the goal before trying other nodes. But this can also lead to significantly slower searches if the final path contains directions that lead away from the goal.

The above algorithm gives us the optimal path for a given source and destination. Once, the path is obtained, the pedestrian moves along the path from source to destination and timestamps are collected every 5 minutes.

The pseudo-code for the A* algorithm is provided in the appendix.

Data Collection

As we did not have real user based GPS logs, we had to generate data ourselves. In order to collect data, we ran simulation on a virtual city with 100 pedestrians. Below is the format of data collected after running it for 24 days.



A	B	C	D	E	F	G	H
Day	Pedestrian	Time	Minutes	xpos	ypos	FemtoCell	
1	100	6:40	400	0	10	0	
1	100	8:50	530	2	22	0	
1	100	8:55	535	1	23	0	
1	100	10:20	620	20	12	0	
1	99	8:50	530	2	22	0	
1	99	8:55	535	1	23	0	
1	98	10:20	620	20	12	0	
1	97	22:05	1325	25	6	0	
1	97	22:10	1330	25	5	0	
1	99	14:45	885	31	2	0	
1	99	14:50	890	32	3	0	
1	100	7:35	455	11	10	2	
1	100	7:40	460	12	10	0	
1	97	19:00	1140	4	18	0	
1	97	19:05	1145	3	19	0	
1	96	0:40	40	18	0	0	
1	96	0:45	45	18	1	0	
1	96	0:50	50	18	2	0	
1	100	2:30	150	30	18	0	
1	100	2:35	155	29	17	0	
1	100	2:40	160	29	16	0	
1	100	2:45	165	28	15	0	
1	100	2:50	170	28	14	0	
1	100	2:55	175	27	13	0	

Each of the above column is explained below:

Column	Description
Day	The number of day for which the execution is running. Ranges from 1-24
Pedestrian	The pedestrian id. Ranges from 1-100
Time	The recorded timestamp for the pedestrian

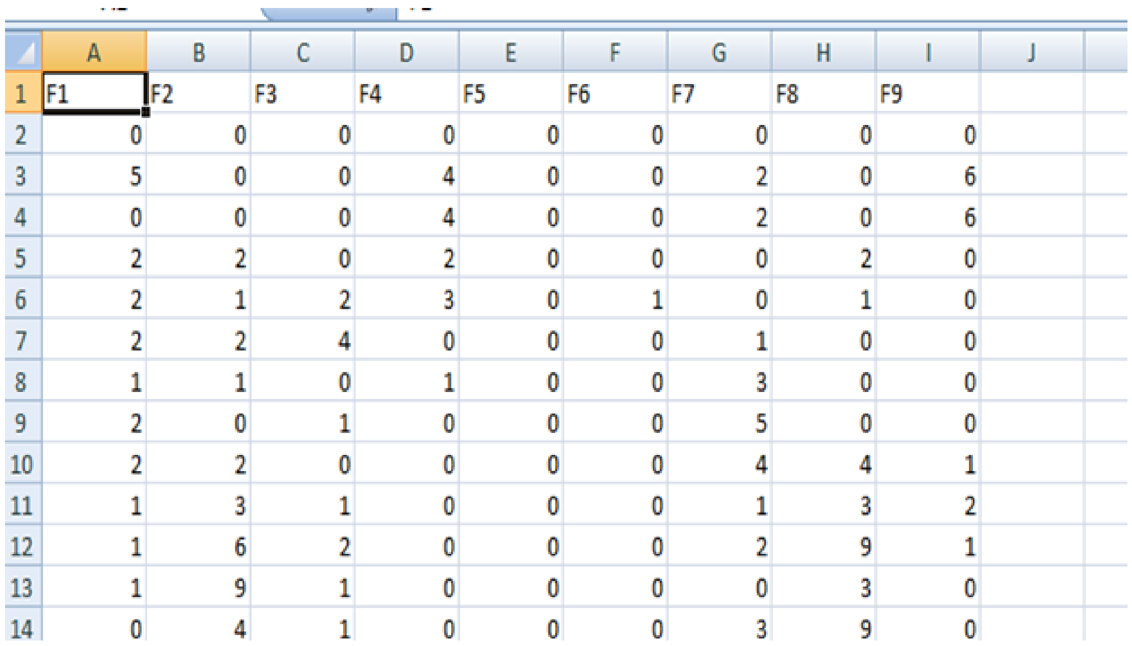
Minutes	Time converted to minute format
xpos, ypos	The location of the pedestrian on the grid
Femto cell	The femto cell number near which the pedestrian is.

Consider the first row, it means that on day 1, pedestrian number 100 was at pos(0,10) at 6:40 and was associated with Femto cell 0.
There are 9 femto cells (1-9) installed in the city at different locations. Here, femto cell 0 indicates that the pedestrian is not near to any of the femto cells installed in the virtual city.

This way the simulation is ran for 24 days and data is collected. This data gives us the location of each pedestrian every 5 minutes. Using this data, pedestrian density can be extracted and used for bandwidth allocation as explained in following section.

Extraction of Pedestrian density:

Using the data collected as explained above, we can extract pedestrian density near each of the femto cell in the city. We wrote a program that reads the above data and calculates number of pedestrians near each femto cell every 5 minutes. This is done for all the days.



	A	B	C	D	E	F	G	H	I	J
1	F1	F2	F3	F4	F5	F6	F7	F8	F9	
2	0	0	0	0	0	0	0	0	0	
3	5	0	0	4	0	0	2	0	6	
4	0	0	0	4	0	0	2	0	6	
5	2	2	0	2	0	0	0	2	0	
6	2	1	2	3	0	1	0	1	0	
7	2	2	4	0	0	0	1	0	0	
8	1	1	0	1	0	0	3	0	0	
9	2	0	1	0	0	0	5	0	0	
10	2	2	0	0	0	0	4	4	1	
11	1	3	1	0	0	0	1	3	2	
12	1	6	2	0	0	0	2	9	1	
13	1	9	1	0	0	0	0	3	0	
14	0	4	1	0	0	0	3	9	0	

As seen, each of the column above shows number of pedestrians at respective femto cell every 5 minutes. For example, consider Femto cell 1. At 0 minutes there are no pedestrian at F1. At 0:5 there are 5 pedestrians at F1.

The above gives us crowd density for entire day. Hence, we can use this data to analyze the density throughout the day and allocate bandwidth on femto cells.

Each of the above column is time-series data which can be used to predict the crowd density in future. The data is used predict crowd density and its analysis is explained in following sections.

Data Analysis

The time-series data collected above was used to perform analysis to predict the future crowd density. Different time-series forecasting models were applied on the above data. Below we show data analysis for femto cell 1.

For Femto cell 1:

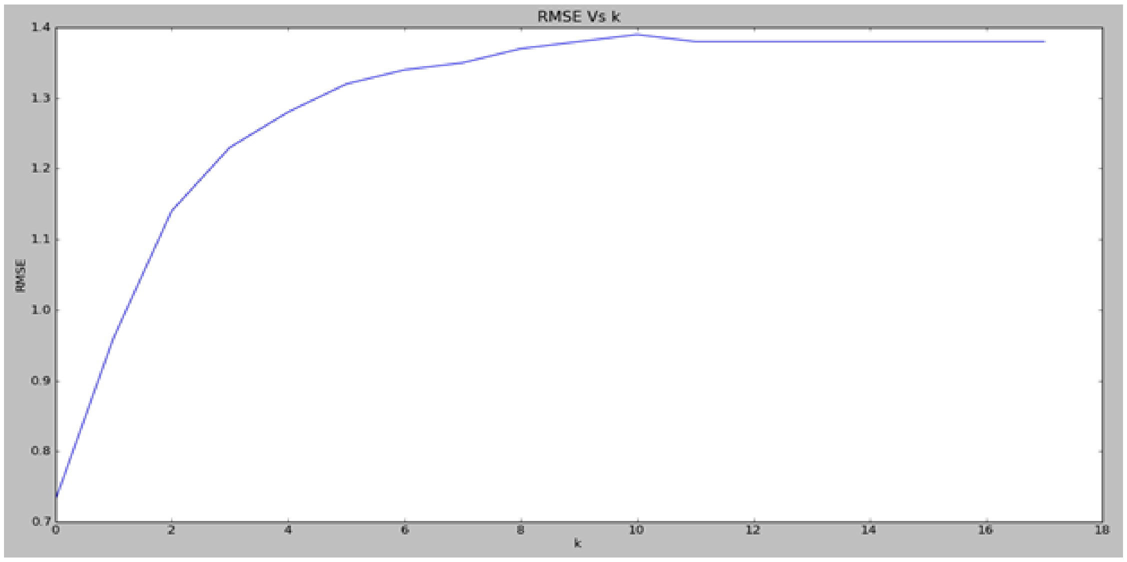
We had collected data for 24 days. However, using the entire data makes visualization of data in plots difficult. So, we used only 5 days data as training data and 6th day data as testing data.

In each of the below tasks, a suitable parameter is first determined and then applied on the training data to calculate the minimum RMSE. Then, the same parameter is used on to predict the test values and compared with the actual test values to determine the correctness of the model.

1. Simple Average model:

In this task, simple moving average model was applied on training data set. Different values of m were used to find Root Mean Square Error.

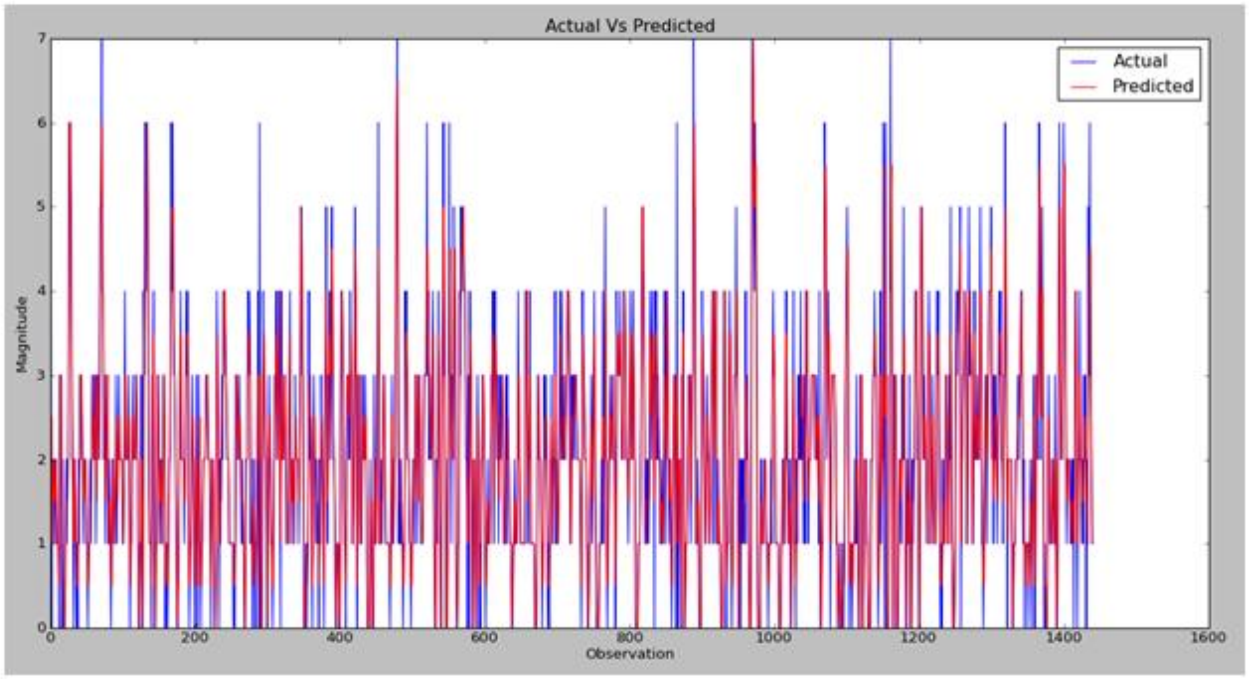
m was varied from 2 to 1500 just to check the variation in RMSE. The variation in RMSE with respect to m is shown below:



As seen, the RMSE initially goes on increasing and then starts decreasing.

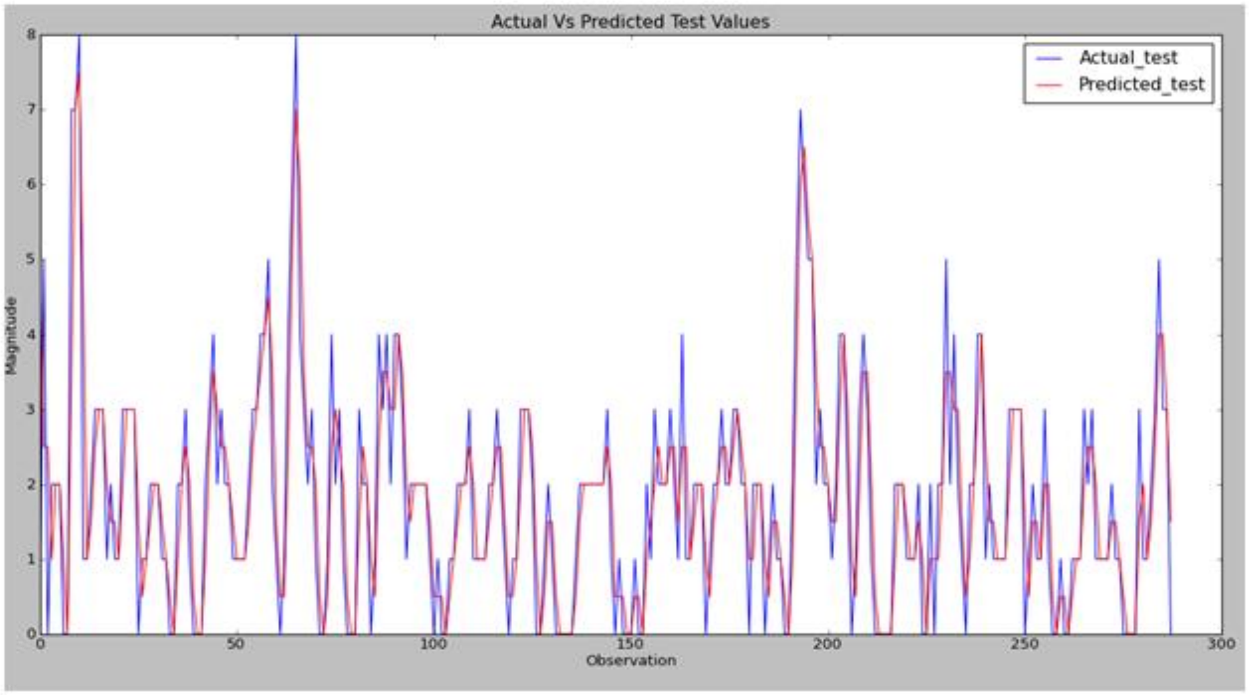
The minimum value of RMSE was achieved for **m = 2**. This was observed for all other femto cells as well. This may be due to the data generated is random. This value of 'm' can be used to predict the values.

Plot of Actual Vs Predicted Training data:



As seen from above figure, the predicted values almost fit the original values. This is because we have used the predicted value with the best value of m which resulted in minimum RMSE.

Plot of Actual Vs Predicted Test data:



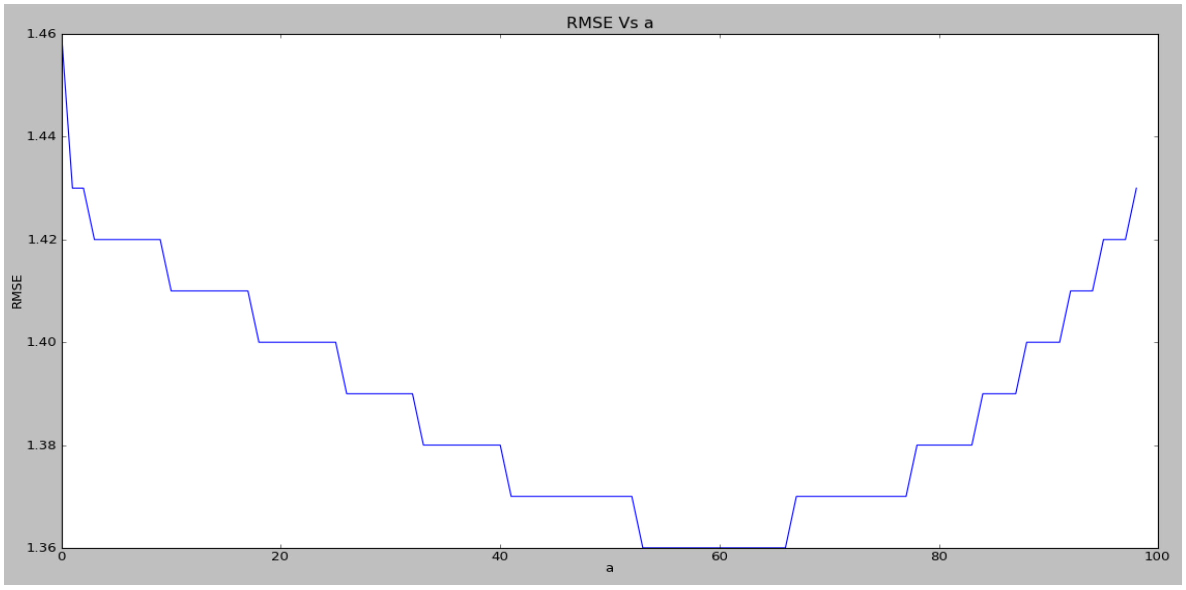
As seen from above figure, the predicted values almost fit the original values of test data. This is because we have used the predicted value with the best value of m which resulted in minimum RMSE for the training data.

2. Exponential smoothing average:

We try to fit an exponential smoothing average model to the training and test data. The model is described by the equation

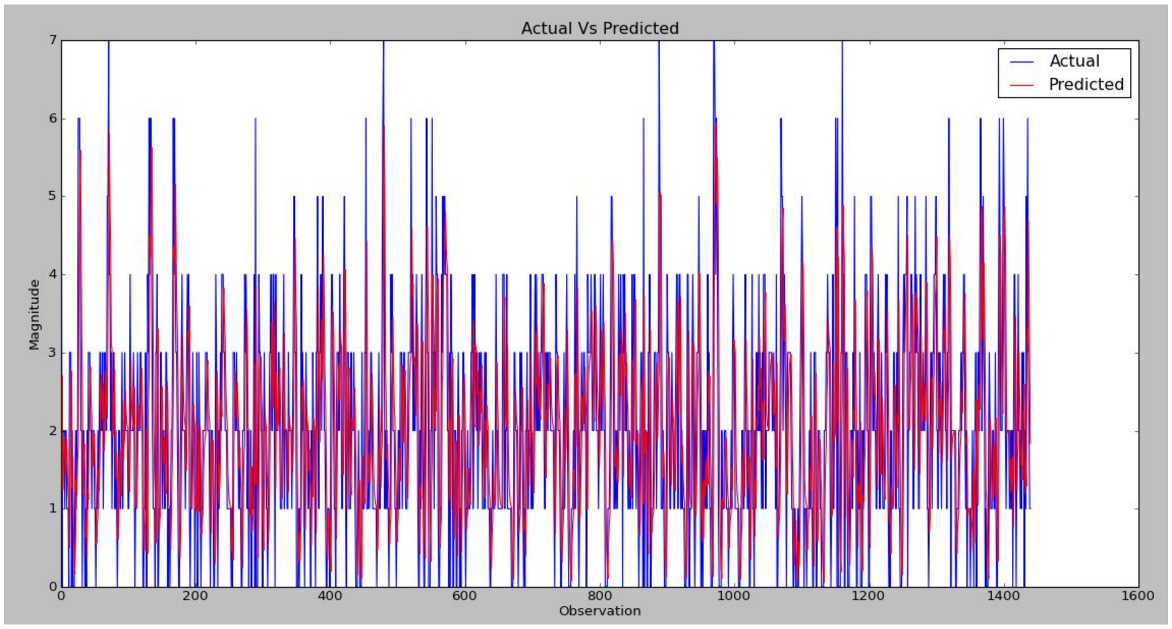
$$S_t = a\, x_{t-1} + (1 - a)\, S_{t-2}$$

Where ‘a’ is the smoothing factor. In order to fit the model we first have to determine the ‘a’ which is carried out by plotting different values of ‘a’ against the obtained RMSE and choosing the value of ‘a’ for which RMSE is the lowest.

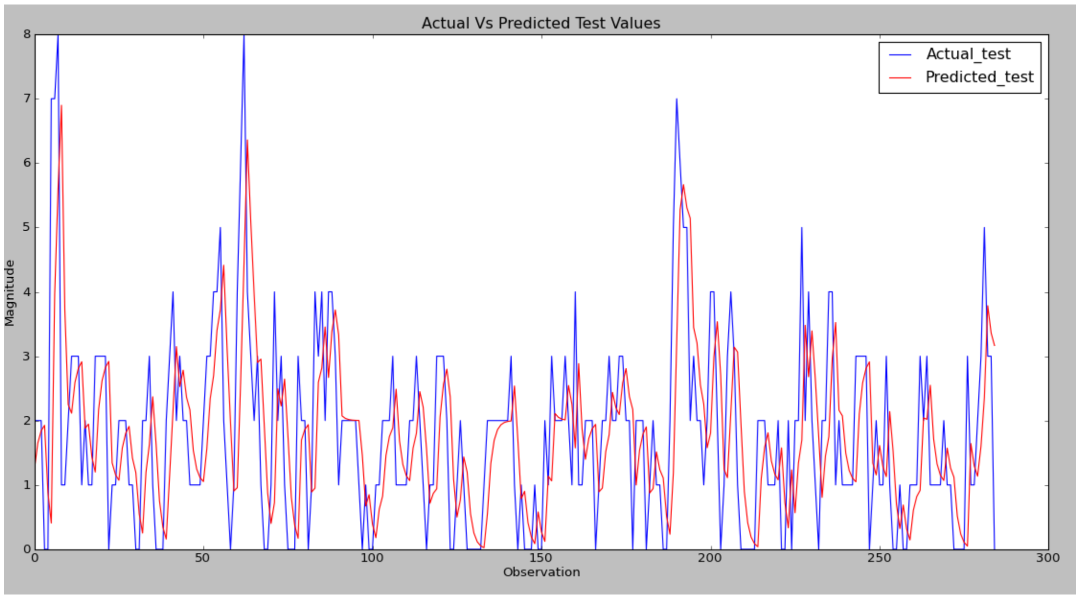


From the above plot ‘a’ turns out to be 0.54. We then proceed to fit the model for the training data which contains 1440 data points (5 days of data taken at 5 minute intervals). The plot is shown below.

Plot of Actual Vs Predicted Training data:



Plot of Actual Vs Predicted Test data:



RMSE obtained is 1.44 for the test data. When compared with the SMA model, the exponential smoothing model does not seem to provide a good fit. We next move on to auto-regressive model.

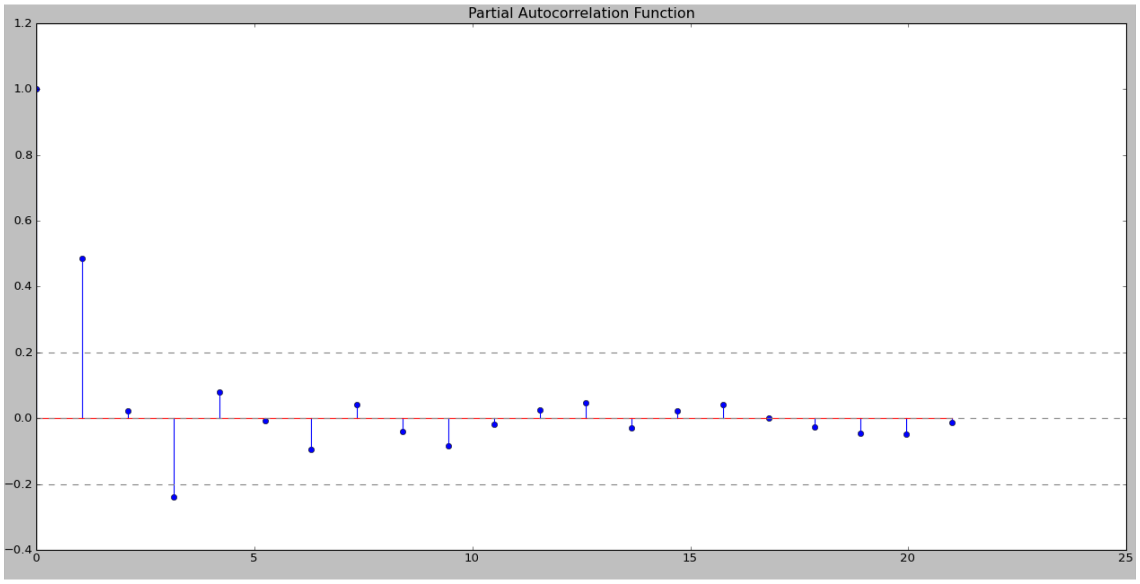
3. AR(p) model

In this task, AR model was applied on training data set. We know that the model can be represented as

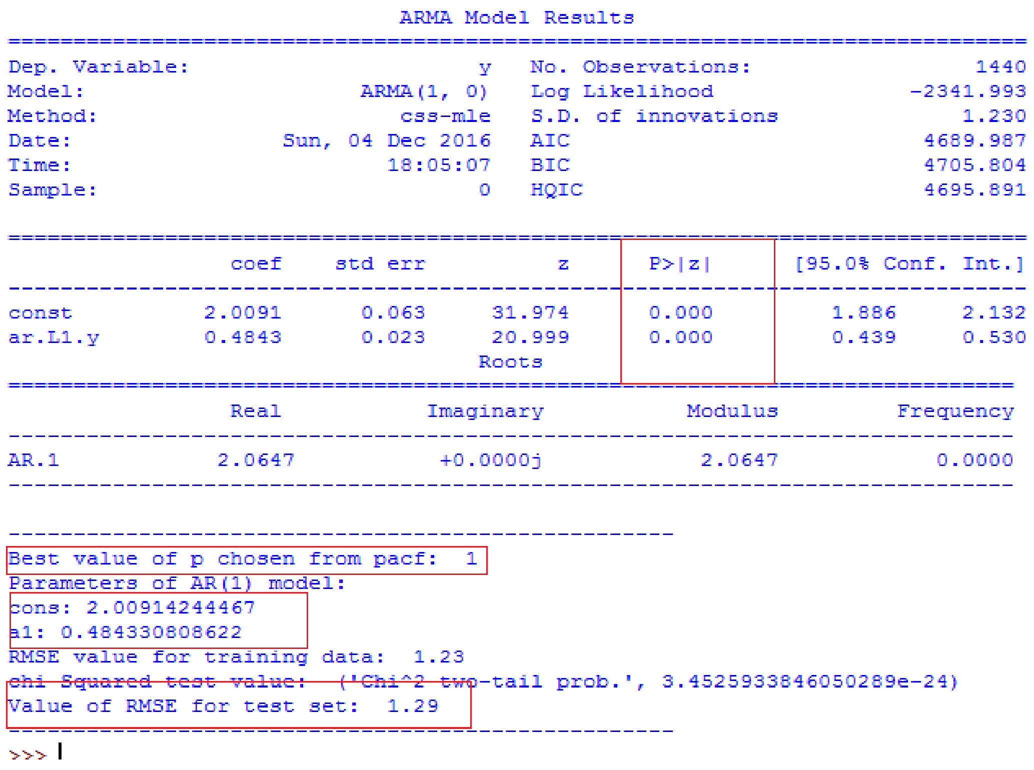
$$X_t = C + a_1 X_{t-1} + a_2 X_{t-2} + \dots + a_p X_{t-p} + \varepsilon_t$$

The value of 'p' is determined from the Partial Autocorrelation function data. The value of p is determined at a lag k past which all other lags of the PACF are zero.

Partial Autocorrelation function Plot:
The plot of PACF values against '*p*' is shown below.



As seen from the above plot, p = 1. For this value of p the parameters of the AR(p) model, δ and a_1, is estimated as shown in the below table.

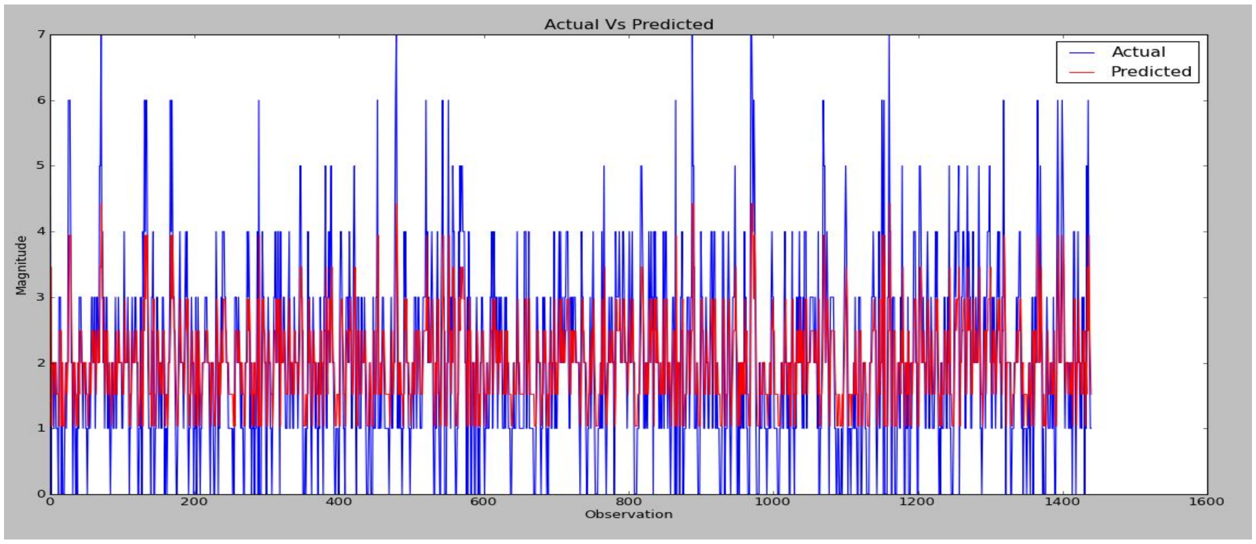


```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1440
Model:                     ARMA(1, 0)   Log Likelihood               -2341.993
Method:                       css-mle   S.D. of innovations              1.230
Date:                Sun, 04 Dec 2016   AIC                           4689.987
Time:                        18:05:07   BIC                           4705.804
Sample:                             0   HQIC                          4695.891

==============================================================================
                 coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          2.0091      0.063     31.974      0.000         1.886     2.132
ar.L1.y        0.4843      0.023     20.999      0.000         0.439     0.530
                                    Roots
=============================================================================
                 Real           Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            2.0647           +0.0000j            2.0647            0.0000
-----------------------------------------------------------------------------

---------------------------------------------------
Best value of p chosen from pacf:  1
Parameters of AR(1) model:
cons: 2.00914244467
a1: 0.484330808622
RMSE value for training data:  1.23
chi Squared test value:  ('Chi^2 two-tail prob.', 3.4525933846050289e-24)
Value of RMSE for test set:  1.29
---------------------------------------------------
>>> 
```

The above results were obtained. As seen the parameters are highlighted above.

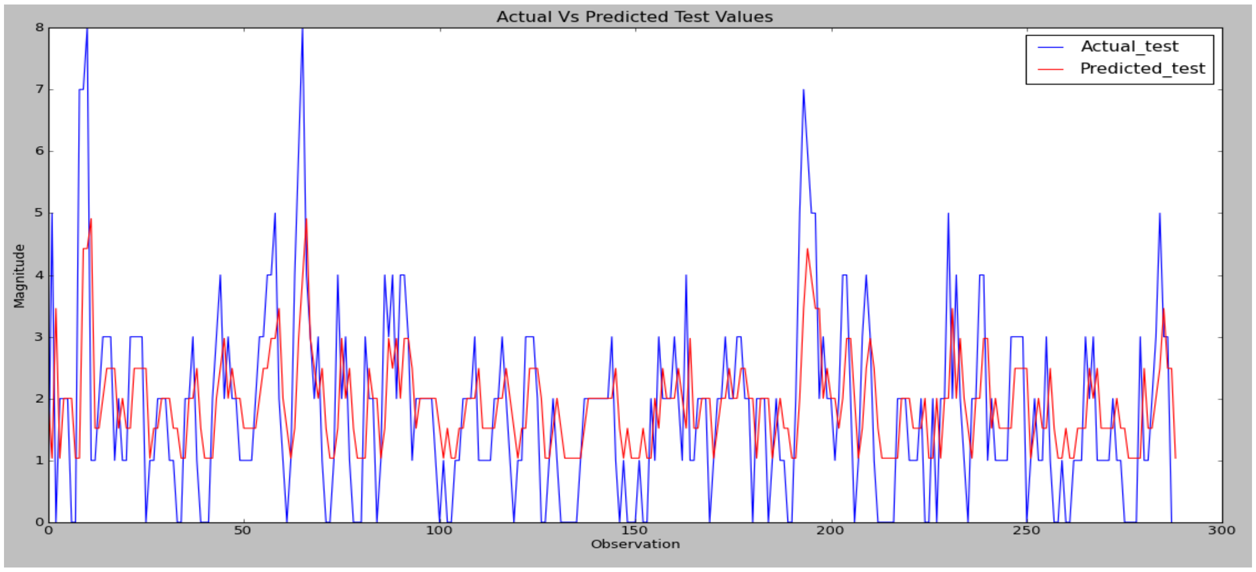
The P-value in the above table tells that the predicted values of the parameters of the model are significant. Using these parameters, values are predicted for the training set and plotted against the original values.

Plot of Actual Vs Predicted Training data:



As seen from above figure, the predicted values averagely fit the original values. It is good fit compared to exponential smoothing model but not as good as SMA model.

Plot of Actual Vs Predicted Test data:



As seen from above figure, the predicted values averagely fit the original values of test data. However, there can be seen gaps between actual and predicted data.

This is not a very good fit as compared to the SMA and exponential models. RMSE value for the test data set is 1.29

Conclusion:

Comparison between above three models:

Model Name	Best value	RMSE for training data	RMSE for test data
Simple average	m = 2	0.73	0.79
Exponential smoothing	a = 0.54	1.36	1.44
AR (p)	p = 1	1.23	1.29

All the 3 models give an approximate fit to the actual data. However, Exponential Smoothing model gives high value for RMSE for both training and test data. Hence, it is not suitable prediction model for this data.

AR(1) model gives lower RMSE compared to Exponential smoothing model, however the AIC and BIC value for this model are high which is not good for predictions.

The simple moving average model provides least RMSE for both training and test data compared to the other two models. It is almost half of what AR(p) model provided and one-fourth of what Exponential smoothing model provided. The lower the RMSE value better is the model for prediction.

Hence, Simple Moving Average model is best model for prediction for the given time series. Hence SMA is used to predict the density of pedestrians around the femtocells and allocate bandwidth accordingly.

Future scope:

This project has presented an idea of allocation of bandwidth in femtocells by simulating movement of pedestrians in virtual city.

Our idea/model could be enhanced by following features:

Time based path and destination selection:

In the morning a pedestrian might travel from his home to office, but in the afternoon he/she might travel from his office to a restaurant or other place. This information can be used to predict the density and allocate bandwidth. Also the we have assumed a the system always has pedestrians walking in the city which is not true in real case. At night time, the density of pedestrians in the city should be less. This can be achieved using putting a bias on inter arrival time when generating pedestrians.

Retrieve information from GPS log of real pedestrians

Pedestrians can carry a GPS device from which actual GPS coordinates can be retrieved as they move along which can give an accurate prediction of the density of people around femtocells.

Prediction using HMM model:

The prediction of the pedestrian's destination can be achieved using HMM models as described in [2] or a trajectory history based probabilistic model as described in [5]

References

1. V. Chandrasekhar, J. G. Andrews, A. Gatherer. Femtocell networks: A survey. *IEEE Communications Magazine*. 2008;46(9):59-67

2. Trip destination prediction based on past GPS log using a Hidden Markov Model J.A. Alvarez-Garciaa, J.A. Ortegaa, L. Gonzalez-Abrilb, F. Velascob, a Computer Languages and Systems Dept., University of Seville, 41012 Seville, Spain, Applied Economics I Dept., University of Seville, 41018 Seville, Spain

3. http://mcsp.wartburg.edu/zelle/python/graphics/graphics.pdf

4. mcsp.wartburg.edu/zelle/**python**/**graphics**.py

5. A personal route prediction system based on trajectory data mining. Ling Chen a, Mingqi Lv a , Qian Ye a , Gencai Chen a , John Woodward b.College of Computer Science, Zhejiang University, 38 Zheda Road, Hangzhou 310027, PR China.The University of Nottingham Ningbo China, Ningbo 315100, PR China

6. Path Finding and Collision Avoidance in Crowd Simulation by Cherif Foudil1, Djedi Noureddine1, Cedric Sanza2 and Yves Duthen2 , Journal of Computing and Information Technology - CIT 17, 2009, 3, 217–228 doi:10.2498 /cit.1000873

Appendix

1. Code for generating the simulation is present at github.

https://github.com/NeetishPathak/Crowd_simulation.git)

2. Pseudo code of A* algorithm:

The following pseudo-code describes the A* algorithm:

```
function A*(start, goal)
    // The set of nodes already evaluated.
    closedSet := {}
    // The set of currently discovered nodes still to be evaluated.
    // Initially, only the start node is known.
    openSet := {start}
    // For each node, which node it can most efficiently be reached from.
    // If a node can be reached from many nodes, cameFrom will eventually contain the
    // most efficient previous step.
    cameFrom := the empty map

    // For each node, the cost of getting from the start node to that node.
    gScore := map with default value of Infinity
    // The cost of going from start to start is zero.
    gScore[start] := 0
    // For each node, the total cost of getting from the start node to the goal
    // by passing by that node. That value is partly known, partly heuristic.
    fScore := map with default value of Infinity
    // For the first node, that value is completely heuristic.
    fScore[start] := heuristic_cost_estimate(start, goal)

    while openSet is not empty
        current := the node in openSet having the lowest fScore[] value
        if current = goal
```

```
            return reconstruct_path(cameFrom, current)

        openSet.Remove(current)
        closedSet.Add(current)
        **for** each neighbor **of** current
            **if** neighbor **in** closedSet
                **continue**                    *// Ignore the neighbor which is already evaluated.*
            *// The distance from start to a neighbor*
            tentative_gScore := gScore[current] + dist_between(current, neighbor)
            **if** neighbor **not in** openSet    *// Discover a new node*
                openSet.Add(neighbor)
            **else if** tentative_gScore >= gScore[neighbor]
                **continue**                    *// This is not a better path.*

            *// This path is the best until now. Record it!*
            cameFrom[neighbor] := current
            gScore[neighbor] := tentative_gScore
            fScore[neighbor] := gScore[neighbor] + heuristic_cost_estimate(neighbor, goal)

    return failure

**function** reconstruct_path(cameFrom, current)
    total_path := [current]
    **while** current **in** cameFrom.Keys:
        current := cameFrom[current]
        total_path.append(current)
    return total_path
```