# Detecting Activities at Metro Stations Using Smartphone Sensors

Aanchal Mongia
IIIT Delhi
New Delhi, India
aanchal15001@iiitd.ac.in

Venkata M. V. Gunturi
IIT Ropar
Rupnagar, India
gunturi@iitrpr.ac.in

Vinayak Naik
IIIT Delhi
New Delhi, India
naik@iiitd.ac.in

***Abstract*—Our paper aims to build a classification-model which delineates the typical motion-related activities performed at a metro station using smart phone sensors. We focus on typical movements, such as climbing the stairs or moving in the lift, waiting at security, waiting at the turnstile to check out and, moving on platform while waiting for a train. Such a classifier estimates crowd levels in a metro-station (and metro trains in an indirect sense), thereby adding towards the vision of efficient metro travel. However, building such a classification-model is challenging due to non-trivial decision boundaries among the classes of interest. Our experiments revealed that the best accuracy that a traditional multi-class classifier could obtain was $0.58$, for a four-class classifier. To this end, we proposed a hierarchical approach of classification which divides the multi-class problem at hand into a set of manageable two-class classification problems. These two-class classifiers are then put together, in a hierarchy, to give an end-to-end solution which takes sensors values from the phone and predicts the class of motion being observed. Our model obtained an accuracy of around $0.75$, a significantly higher value, on the real-data collected at Delhi Metro stations. The same classifiers can potentially be applied to detect crowd levels at train stations and bus depots, which will make transportation efficient in smart cities.**

***Index Terms*—Mobile Sensing, Smart Cities, and Machine Learning**

## I. Introduction

With the ever increasing proliferation of mobile devices with advanced sensing capabilities, the approach of participatory sensing has become an important source of data collection. Google Maps collects its traffic data from its mobile phone apps. The crowd-sourced applications are becoming an integral part of our daily lives.

In this paper, we use the concept of participatory sensing to contribute towards building a system, which tracks the typical motion activities of a metro train commuter, such as climbing stairs, waiting in queues, etc. Such a system is likely to find use-cases in the domain of crowd estimation at metro stations (and to some extent in metro trains in a indirect way), train stations, or at bus depots. Thus, eventually leading to the concept of smart-travel in the modern cities.

An intuitive approach for measuring the level of crowd could be either through complex image processing algorithms on Close Circuit TV (CCTV) footages, or by leveraging RFID (radio frequency identification) sensor data logged at metro stations. However, the lack of availability of such data for third party apps is a challenge. Also, even if one assumes that the data for the former approach becomes accessible, the image data obtained would not only be restricted to a small line-of-sight area, but will also be a breach to people's privacy, because through this data a commuter could be observed on a daily basis; tracking the route he (or she) typical takes without his (or her) consent. In this context, one may note that public administration offices in India have implemented several policies on CCTV data. For instance, state government of Maharashtra, India has declared that data obtained from CCTV is classified [1], which means that it cannot be obtained even under the RTI Act. Similarly, Delhi Metro preserves the CCTV data only for 7 days after which it is destroyed [2].

In contrast to image processing based approaches, we propose to approach this problem through sensor data collected through a crowd-sourced mobile application that is deployed on smartphones carried by volunteers. We believe that our approach is better than previously stated solution through CCTVs because it not only overcomes the stated limitation of privacy concerns, but also, has an added advantage that even if very few people agreed to contribute to our data set, the prediction model would still give good results. This is because each commuter will go through all stages of awaiting at the stations. Smartphone users can be encouraged to participate in data contribution and install the application by giving them incentives.

As a first step towards solving the crowd estimation problem, in this paper, we propose a motion classification system which uses data from the smartphones sensors. The goal is to develop a model which can predict type of movement the commuter is exhibiting. We formalize this as a multi-class classification problem, into classes corresponding to waiting in queues, climbing down on stairs or elevator, etc. Presence of some of these classes could be considered as a signal of crowd at the station. For instance, occurrence of "waiting in queues" after "climbing down stairs" could not only imply that the station is crowded, but also hint that the metro journey itself was quite congested. Figure 1 illustrates some of these classes at Delhi Metro Stations.

Following are the key aspects of this paper:

- We propose an approach to accurately predict the typical motion activities in a metro station. These include climbing up/down the stairs (or use an elevator), moving in a queue, standing still and generally walking on the

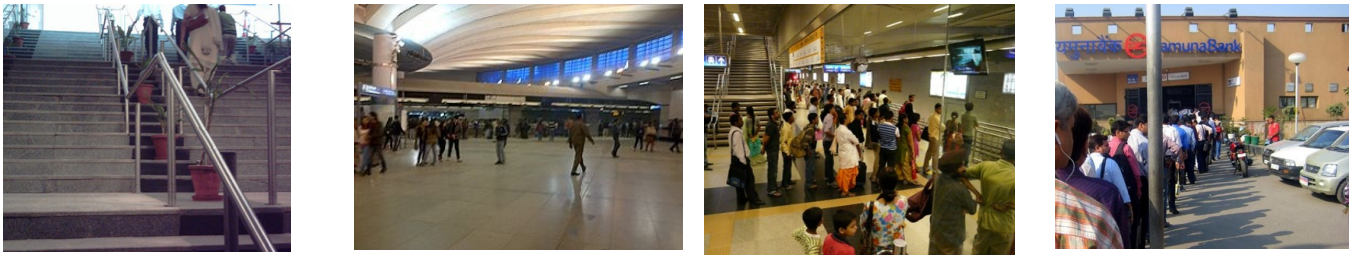

Fig. 1: Typical motion-activities corresponding to our target classes *Climbing*, *Moving*, *Stationary*, and *Queuing*

platform. This is modelled as a multi-class classification problem with four classes.

- Our approach breaks this four-class classification into a series of two-class classification problems. These two-class classifiers are arranged in a multi-level hierarchy to provide an end-to-end prediction. At the first of our hierarchy, our model catches the instances of climbing up or down (either by stairs or elevator). If given test point filters through, then at the second level, we catch instances of "generally walking on platform." Following this, at the third level, we classify the test point as either "moving in a queue" or "standing still."
- At each level, we explored multiple options for classifiers and window sizes. In our model, we first take a time-series of sensor values and represent them using a set of features (details in Section IV).
- For classifiers, we explored Nearest Neighbor algorithm, Regularized Logistic Regression, Neural networks, and Support Vector Machines (SVM).
- Overall, our results with ten-fold cross validation indicate that a hierarchy with SVM at level two and Neural Networks at level three work the best. And, the optimum performance is obtained at window size of ten seconds (corresponds to twenty time-units in our implementation).

The rest of the paper is organized as follows. Section II discusses related work in the area of Crowd Estimation, Classification using data from sensors, and Activity Recognition using smart phones. Section III formally describes our problem statement. The approach and methodology followed are discussed in Section IV. This is followed by the results of our experiments conducted and the evaluation strategy in the Section V. Section VI finally concludes the paper.

## II. Related Work

The work pertaining to traditional solutions to crowd level estimation include video (or image) processing and RF-based approaches. To this end, Li et al [3] proposed a method to address the problem of estimating the number of people in surveillance scenes with people gathering and waiting by combining a MID (Mosaic Image Difference) based foreground segmentation algorithm and a HOG (Histograms of Oriented Gradients) based head-shoulder detection algorithm to provide an accurate estimate of people count. [4], [5] extract information from video surveillance sequences of the motion vectors of salient points and use statistical analysis to obtain statistical measures on data for crowd detection. But, video/image based approaches have limited line of sight, requires the environment to be deployed with video infrastructures and are a breach to people's privacy as humans can be tracked without their consent. And moreover, accessibility of video/image data is an issue at the place of our interest, i.e. metro stations.

An interesting and reasonably accurate solution for crowd-counting based on audio tones was presented by Kannan et al [6]. They efficiently leveraged the microphones and speaker on phones to use audio tones which are barely audible to humans. This method has limited applicability in our case because of the following two things. First, speakers on some mobile phones (e.g. HTC Desire) introduce noise in the transmitted frequency used by them, which affects the accuracy of their algorithm. Certain mobile phones (e.g. Nexus One) are unable to sample frequencies above the threshold frequency they had set (10 KHz) and so such phones cannot be used. Secondly, noise does not directly co-relates with crowd.

The above works attempt to solve the problem of crowd density estimation in some way but none of them perform activity classification on data collected from a smartphone sensor on a subway or metro for contributing to a possible solution for crowd estimation. Vij et al [7] solved the problem of using accelerometer in smartphones to detect mobility activities (whether user is at a metro train station or in a metro train) during the metro train travel using various machine learning classification techniques with two different data representations, statistical features and ECDF-based features. For this binary classification problem, they claim that the ECDF-based feature representation suits better with the use of an ensemble of a classification model and achieve a precision of 0.98. [8] uses accelerometer, GPS and other sensors, and gives an activity classification algorithm for determining whether or not the user is riding in a vehicle. They propose a memory and time-efficient route matching algorithm for determining whether the user is in a bus or another vehicle and a method for tracking underground vehicles. Modelling transportation mode detection as a multi-class classification problem is common in many works. [9] and [10] use smart device sensors to classify transportation modes; the former employs GPS and accelerometer data from smart devices to classify outdoor transportation modes (which include walking, bicycling, motorized transport and stationary

state) using support vector machines, while the latter explores indoor transportation mode detection in a hierarchical fashion using accelerometer and WiFi measurements collected through smartphones using four different classifiers (C4.5 decision tree, support vector machines and random forest).

Sankaran et al [11] presented an approach for context detection using a smartphone barometer. They just determine whether the user is riding on the subway (VEHICLE state) or is involved in any other activity (IDLE or WALKING state). We, on the other hand, detect other activities when the user is not riding in the subway by segregating instances of each micro-activity.

A lot of research has been done in the domain of Human Activity Recognition (HAR) using sensor data. Some works use the data collected from sensors which are mounted on the body part of the subject (wearable sensors), while others use sensors which are embedded in the user's smartphone. The latter is a favourable approach for data collection as it avoids the discomfort caused to the subject by the body-mounted sensors. HAR using mobile phones has been effectively exploited for daily activity monitoring [12] and user mobility identification [13].

Su et al [14] presented a study of recent advances in activity recognition with the smartphones' sensors. They gave a review of the core data mining techniques behind the main stream activity recognition algorithms and showed a variety of real applications of the same. Another work by Mitchell et al [15] presented a framework that allows the automatic identification of sporting activities using smartphone accelerometers by extracting discriminative informational features using the Discrete Wavelet Transform (DWT). But, the approach has a disadvantage that, while using the DWT, when extracting features from signal, there is no widely accepted method of picking the most suitable mother wavelet for a particular application.

Apart from these, a Patient Monitoring Application was introduced in [16] which enabled activity recognition for patients using non-obtrusive devices through investigation of a movement recognition approach using a smartphone with a built-in accelerometer.

## III. Problem Statement

Before mentioning the problem, we define classes.

### A. Motion Classes focused

Following motion classes are focused in our model:

1) **Climbing**- This class corresponds to when the user is climbing up or down the stairs in a metro station. It also includes cases when the user is moving up or down in a lift.
2) **Moving**- In this case, the user is casually walking on a platform, probably waiting for a train. It also includes cases when the user is either sprinting or walking briskly.
3) **Stationary**- This class corresponds to instances during which the user is absolutely still, i.e., either he is sitting or standing or waiting on platform for the metro train to arrive.
4) **Queuing**- In this case, the user is slowly moving in a queue at a metro station, probably waiting to buy a ticket, clear security, or check-out at the turnstile.

### B. Problem Definition

**Input:** A time series of smartphone sensor readings (barometer and accelerometer) collected at metro stations. The readings are recorded at a temporal granularity of $0.5$ seconds. This means that readings at time indices $1$ through $10$ imply $5$ seconds of data. These readings are divided into a set of overlapping windows, where two consecutive windows, each of length $n$ time indices overlap for $n-1$ time-slots. Each window forms a data-point for our classification model.
**Objective**: To learn a classification model which can effectively classify a given data point (a window across $n$ time-slots, i.e., $n/2$ seconds) into one of the previously described four motion-classes *Climbing*, *Moving*, *Stationary*, and *Queuing*.

In other words, given raw data from the accelerometer and barometer sensors over a window of time-points; the goal is to determine whether the user of the smartphone is climbing, moving, stationary, or in a queue during that window.

Results from this classification model help estimate the crowd-level in an indirect sense. For instance, if we notice that users typically have to be in a queue for a long time after entering the station (determined through geo-fencing) at 9:00am, then probably this metro station faces crowd at this time. Similarly, if a good number of users experience a long queue after getting down at a metro station at say 10:00am, then it could mean the trains arriving at that station at 10:00am may be crowded.

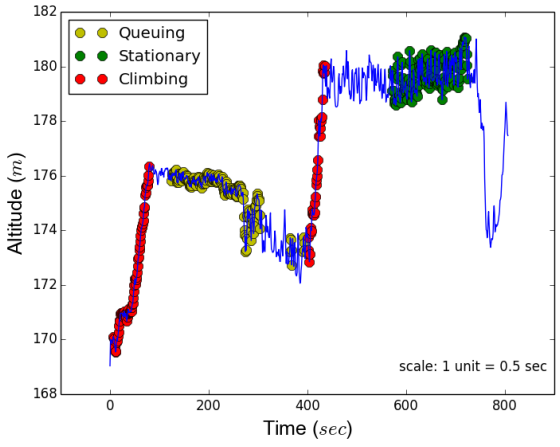


Fig. 2: Sample plot showing Altitude values at metro station against time. It depicts the usefulness of barometer in detecting *Climbing* class (red) (best in color).

**Sample data**: Figure 2 illustrates classes on the altitude values (derived from barometer sensor) against Time. Here, 2 values correspond to 1 second of data. The Figure shows

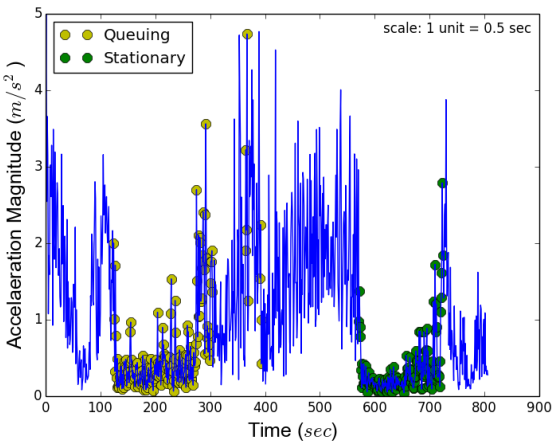


Fig. 3: Sample plot showing Linear acceleration magnitude values at metro station against time. It shows the the usefulness of accelerometer sensor in delineating *Queuing* (yellow), *Stationary* (green) , and *Moving* (others) classes (best in color).

that barometer values are most useful for the climbing class (shown using red points). It does not help other classes, such as *Queuing* (yellow points), *Stationary* (green points), and *Moving* (others).

Figure 3 depicts a sample data set corresponding to the magnitude of linear acceleration readings taken at a metro station through a plot of acceleration magnitude values against Time. Similar to previous plot, 2 values correspond to 1 second of data. The Figure shows that this sensor is very useful for delineating *Queuing* class (yellow data points), *Stationary* class (green data points), and *Moving* class (others). One can observe that when the user is moving, the acceleration magnitude shows a lot of variation, with values greater than $2m/s^2$ (approx). Whereas, when the user is standing or is in a queue, the linear acceleration magnitude and its variation is less.

## IV. Proposed Approach

### A. Pre-processing of the data

Raw sensor data readings are collected at a sampling rate of two values per second. In our application, we use data from the accelerometer and barometer. Following text describes our procedure of prepossessing the raw data:

- **Acceleration:** In our application, acceleration is measured along each of the three axes using a linear accelerometer (instead of gravity sensor) which provides us with a three-dimensional vector representing acceleration along each device axis without the gravity component. For each time index, we compute the magnitude ($a$) of the three dimensional linear acceleration vector using the following equation:

$$a = \left(\sqrt{a_x^2 + a_y^2 + a_z^2}\right) \tag{1}$$

- **Pressure** value at each time index is converted to height/altitude in meters by the following equation:

$$h(p, p_o) = 44330 * \left(1 - \left(\frac{p}{p_o}\right)^{\frac{1}{5.255}}\right) \tag{2}$$

Here, $p$ is the atmospheric pressure obtained from the sensor and $p_o$ is the pressure at sea level, which is retrieved from airport databases in the vicinity [17]. After getting the altitude values, these are smoothed (for de-noising) by applying median filter.

After pre-processing the accelerometer and pressure values, the data is quantized into a set of overlapping windows of a suitable length (this parameter is varied in experiments). As mentioned earlier, two consecutive windows of length $n$ time indices, overlap on $n-1$ time indices. Each window becomes a data point in our model. We represent the accelerometer data of a window by computing the max, mean, mode, and standard deviation of all values inside the window. All the altitude values of window are summarized into just one number, 1 or 0. If all the altitude values inside the window are either monotonically increasing or decreasing, then we summarize the data as 1, otherwise 0. It should be noted that this approach would not be plagued by the presence of outliers because any outliers present would be smoothed out during the pre-processing stage itself. For instance, a window with altitude values $20, 21, 22, 2, 24, 25$ is not possible as 2 would be pruned out by the median filter during the pre-processing stage.

### B. Proposed hierarchical model

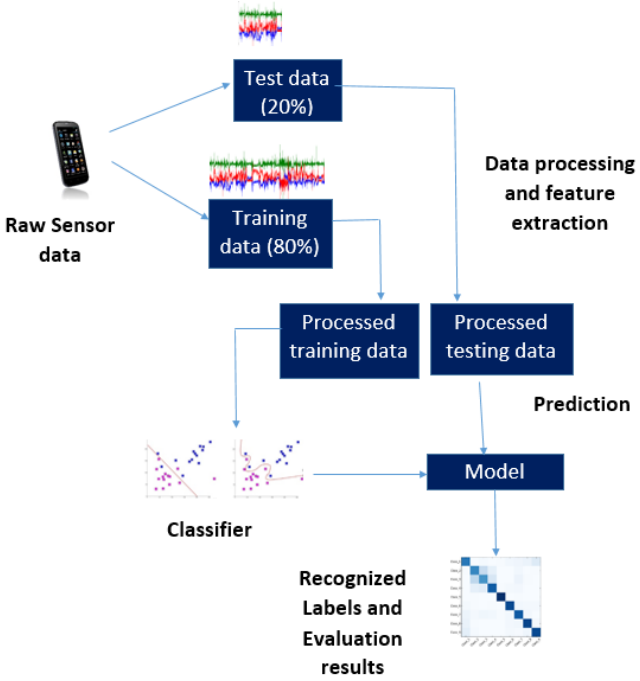


Fig. 4: Overview of processing steps

Figure 4 illustrates an overview of the proposed solution. We first briefly provide the key intuitions used to detect each of the classes. Following this, we provide details on training

our hierarchical model for addressing our stated four-class problem.

- **Climbing**: If all altitude values in a window form a strictly increasing (or decreasing) sequence, then this window probably corresponds to climbing up (or down) activity.
- **Moving**: It is observed that when the user is moving, the acceleration magnitude shows a lot of variation, with values greater than $2m/s^2$ (approx). On the other hand, when the user is standing in a queue (or stationary), the magnitude of acceleration and their variation is less. It is very likely that our features of mean, mode and max would capture this signature in acceleration.
- **Queuing**: This method works on similar lines as that of the moving class with the difference being: much lesser mean and standard deviation of each window.
- **Stationary:** It is observed that when the user is standing, the acceleration magnitude shows the least amount of variation. When the user is in a queue (moving queue), the variation in the magnitude of acceleration is more than what is typically observed during *Stationary* periods.

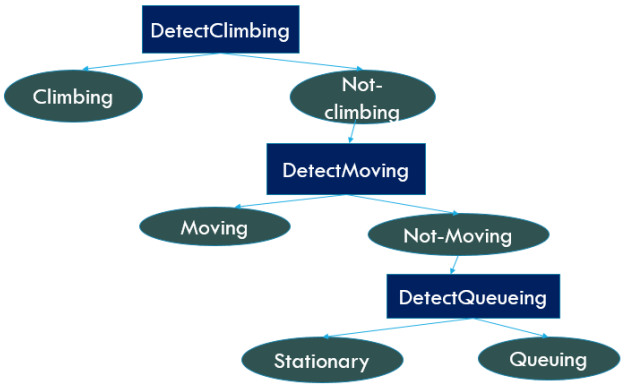


Fig. 5: Hierarchical model design involving tasks at each level to classify data points

As mentioned earlier, we propose to build a hierarchical model to discern our four classes. In our hierarchical model, the problem of discerning the four classes is broken down into a series of two-class problems. These two-class classification problems are arranged in a hierarchy to create a model which predicts the class of a given test data-point (a window of sensor values after pre-processing). Figure 5 illustrates our proposed hierarchy. We now detail our procedure to train this model.

In the following discussion, assume that we have a set $S$ of windows of length $n$ as data-points (after pre-processing) for training. These windows are labeled (in one of the four classes). As stated earlier, the accelerometer values of this data-point are represented using features mean, max, mode, and standard deviation. All altitude values in the window are summarized to just binary values (1 or 0). Following a step-wise procedure of training the data:

**Step 1 (Training at Level 1):** The labeled training data in the set $S$ is divided into two parts. In one part we collect all the *not-Climbing* data-points, which are essentially all the windows which have a label of *Moving*, *Stationary*, or *Queuing*. In the other part, we put windows which have the label *Climbing*. Now taking this as the training data, we learn a classification model between *Climbing* and *not-Climbing*. At this level, we use only the binary or 1/0 summarization of the altitude values of the window for training. In our experiments we explore multiple binary classifiers, such as neural networks, logistic regression, SVM etc., at level 1.

**Step 2 (Training at Level 2):** We again take the labeled training in the set $S$. From this, we pickup a new training set $S'$ which contains windows corresponding to only *Moving*, *Queuing*, and *Stationary* classes. Similar to the step 1, we divide $S'$ into two parts. In one part, we collect all the *not-Moving* data-points, which are essentially all the windows in $S'$ which have a label of *Stationary* or *Queuing*. In the other part, we have windows which have the label *Moving*. Using $S'$ as the training data, we train a classification model between *not-Moving* and *Moving* classes. At this level, we use only the accelerometer features (mean, mode, and max) of the window for training. Similar to step 1, we explore multiple binary classifiers, such as neural networks, logistic regression, SVM etc., at level 2 as well.

**Step 3 (Training at Level 3):** We again take the labeled training in the set $S$. From this, we pickup a new training set $S''$ which contains windows corresponding to only *Queuing* and *Stationary* classes. This is the training set used at level 3. At this stage, using $S''$ as the training data, we learn a classification model between *Stationary* and *Queuing* classes. We use only the accelerometer feature of standard deviation of the window for training. Again, similar to steps 1 and 2, we explore multiple binary classifiers such as neural networks, logistic regression, SVM etc., at level 3 as well.

The combination of the models learned at levels 1, 2, and 3 forms our hierarchical model. Given a test window, this model can map it to one of the four classes, *Climbing* ,*Moving*, *Stationary*, or *Queuing*. The test point will first pass through the model *Climbing*-vs-*not-Climbing* (at level 1). If the test point comes to *not-Climbing*, then we pass it through the model *Moving*-vs-*not-Moving* (at level 2). Again, if the test point falls into *non-Moving*, then we pass it through the *Stationary*-vs-*Queuing* model (at level 3). In the end, all test points are guaranteed to fall in any one of the four classes. Of course, as our also results indicate, there would be some amount of test error.

**Classifiers used:** We have tried the following classifiers at each level.

- K-NN classifier
- Neural Networks (Multilayer Perceptron classifier with 2 hidden layers having 100 and 300 neurones each)
- Regularized Logistic Regression

- Support Vector Machines (SVM) (with RBF kernel)

It should be noted that we took 2 hidden layers in the Multilayer Perceptron classifier because atleast 2 layers are needed in a neural network to learn a decision boundary on non-linear data. Also, after trying various combinations of number of hidden layer nodes (from 1 to 1000 nodes in each layer), we come to a conclusion that this combination of 100 nodes in first hidden layer and 300 nodes in second hidden layer gave good results, with no improvement observed in accuracy if the number of hidden units are increased any further.

### C. Flat model

As an alternative approach for our four-class classification problem, we also explored what we refer as to flat model. In the flat model, we directly learn a multi-class classifier. This model classifies a test data point in a single step instead of three steps (as done in our hierarchical model). For the flat model, each window is represented using the following 5 features: 0/1 summarization of the altitude values and; mean, mode, max, and standard deviation of the accelerometer values.

For the flat model, we explored K nearest neighbor classifier, Regularized Logistic Regression, Multilayer Perception, SVM (with RBF kernel), and Adaboost (with decision tree as its base estimator). Note that SVM is typically used as a binary classifier. For our problem, SVM was generalized to multi-class problem by using the one-vs-all strategy. In this strategy, we learn following four different decision boundaries using four different instances of SVM: *Climbing*-vs-*Others*, *Moving*-vs-*Others*, *Stationary*-vs-*Others*, and *Queuing*-vs-*Others*. Based on these four boundaries, we predict an appropriate label of the given test point.

## V. Evaluation

This section gives the details on the experiments conducted to evaluate the performance of our proposed models.

**Dataset used** We used real data for our experiments. We developed an Android mobile application to collect the data. In this application, there were buttons to record the ground truth. For instance, the application had buttons "enter queue" and "exit queue". Note that such a UI in the mobile application would be needed only while training the model. After a model is built, it can be launched through another mobile app (or a website) which shows a real-time map of metro congestion based on the raw sensor data coming from volunteers. This app will not have any buttons used for data collection. We collected data through a Samsung Galaxy phone. For our experiments, we collected data at stations along the blue and purple lines of Delhi Metro network [1]. Overall, our data had raw sensor values for 13217 time indices. Given that our application sampled data once every half a second, 13217 time indices correspond to about two hours of data at spent at metro stations. Note that these 13217 time indices do not include the amount of time spent while travelling between metro stations.

After pre-processing, data was divided into a set of overlapping windows. As mentioned earlier, each window forms a data-point in our model. In our experiments, we varied the length of the windows as well. As one can imagine, as we vary the window length, the number of resulting "data-points" (and thus #class instances) would also change. Table I details the number of data points and #class instances for different window lengths.

| Window Length | #Moving | #Queuing | #Stationary | #Climbing | Total #data-points |
|---|---|---|---|---|---|
| 5 (2.5 secs) | 6216 | 2324 | 2409 | 2258 | 13207 |
| 15 (7.5 secs) | 6216 | 2324 | 2409 | 2238 | 13187 |
| 25 (12.5 secs) | 6216 | 2324 | 2409 | 2218 | 13167 |
| 35 (17.5 secs) | 6216 | 2324 | 2409 | 2198 | 13147 |
| 45 (23.5 secs) | 6216 | 2324 | 2409 | 2178 | 13127 |

TABLE I: Dataset in terms of window lengths. Table also shows the number of instances of each class

The windows in the dataset were randomly divided into two parts in $80:20$ proportion. $80\%$ of data was used as training, while the remaining $20\%$ was used for testing the data.

**Parameters varied** We evaluated our proposed approach (Hierarchical model) and the Flat model by varying following parameters:

- Window size: To find out the ideal quantization of the raw sensor data (ideal window size from which features need to extracted), we varied the window size from 5 to 50 (at a quanta of 5 values).
- Classifiers for the Hierarchical model: We have evaluated each of the three levels using K-NN classifier, Multilayer perceptron, Regularized Logistic regression, and Support Vector Machines (with RBF kernel).
- Classifiers for the Flat model: Support Vector Machines generalized for mulit-class (using One-vs-All strategy), K-NN classifier, Multi-layer perceptron, and Regularized Logistic regression and AdaBoost (with decision trees as the based classifier).

**Metrics of evaluation** As the parameters were varied, we measured confusion matrices. These matrices were summarized using accuracy and F-measure (as it more stable to class imbalance) in our experimental plots after doing ten-fold cross validation.

### A. Experimental results for the hierarchical model

Figure 6 shows the variation in accuracy values (for the test data) across different window sizes. Figure 7 shows the variation in F-measure. The left most plot in Figure 6 (and Figure 7) shows the results at level 1. At this stage, we have classified the test data as *Climbing* or *non-Climbing*.

[1] www.delhimetrorail.com

(a) Level 1 plot (b) Level 2 plot (c) Level 3 plot

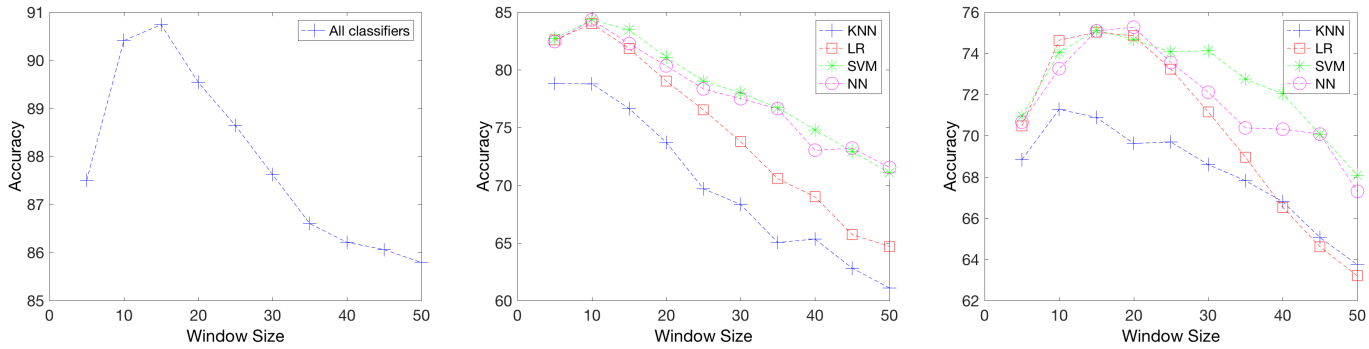


Fig. 6: Variation of Accuracy Scores with different window sizes at each level of hierarchy (with KNN classifier, Regularized Logistic Regression, SVM and Multi-layer perceptron classifier) showing best accuracy at a window size of 20 with any classifier at level 1, SVM at level 2 and MLP classifier at level 3

(a) Level 1 plot (b) Level 2 plot (c) Level 3 plot

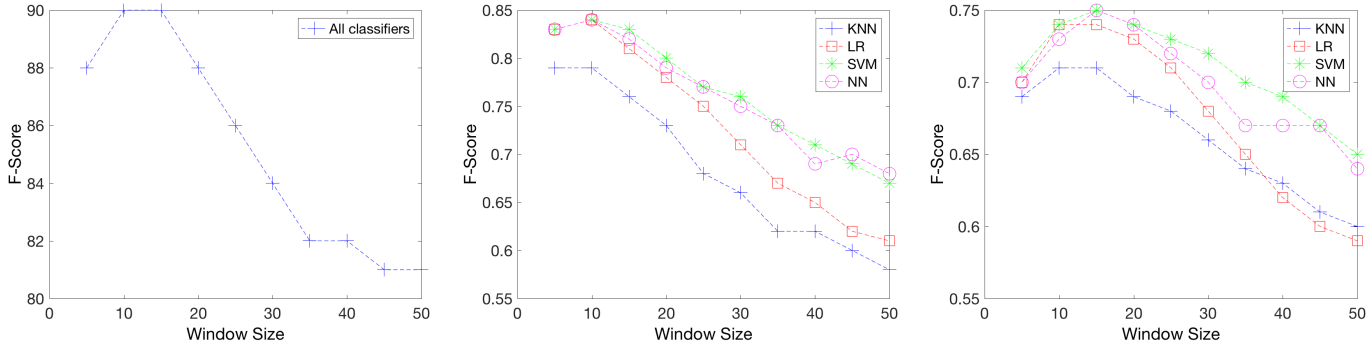


Fig. 7: Variation of F-measure with different window sizes at each level of hierarchy (with KNN classifier, Regularized Logistic Regression, SVM and Multi-layer perceptron classifier) showing best accuracy at a window size of 20 with any classifier at level 1, SVM at level 2 and MLP classifier at level 3

Our results indicated that, at this level, performance of all the classifiers were identical. This is understandable as we used only summarization (summarized to either 1 or 0) of the altitude values to represent the window.

The middle plot in Figure 6 (and Figure 7) illustrates the performance of different classifiers for discerning *Climbing* and *Moving* from *Others*. For each line in this plot, we passed the test data through a model comprising of best-performing-classifier at level 1 (any of the SVM, KNN, LR, etc., as they all performed the same at level 1) and the concerned model at level 2 (SVM, KNN, Logistic Regression, etc.). The accuracy (and the F-measure) values in this plot are derived from a $3\times3$ confusion matrix containing *Climbing Moving* and *Others* classes.

The third plot in Figure 6 (and Figure 7) illustrates the performance of different classifiers for delineating *Climbing*, *Moving*, *Stationary* and *Queuing* classes. For each line in this plot, we passed the test data through a model comprising of best-performer at level 1, best-performer at level 2 (SVM) and the concerned model at level 3 (SVM, KNN, Logistic Regression, etc.). The accuracy (and the F-measure) values in this plot are derived from a $4 \times 4$ confusion matrix containing *Climbing*, *Moving*, *Stationary* and *Queuing* classes.

Our experiments reveal that with the choice of 20 as the window size, the full hierarchical model with any-model at level 1, SVM at level 2 and Multi-Layer Perceptron at level 3, performed the best by attaining an accuracy of 0.75 and a F-measure of 0.74 for our four-class classification. The resultant confusion matrix with accuracy and F-measure is shown in Table II. Here, class 0, 1, 2, and 3 correspond to classes *Moving*, *Queuing*, 'Standing', and *Climbing* (up or down) respectively. Table III shows the Precision, Recall, and F-measure for all classes on the best performing hierarchical model. At this window size (20 data points), the model at level 1 separates the *Climbing* instances with an accuracy of 0.89, the model at level 2 which performs binary classification task between *Moving* and *Not-Moving* instances, has an accuracy of 0.89 and the binary classification model at level 3 differentiates the

stagnant queue scenario from the stationary with an accuracy of 0.84.

TABLE II: **Confusion matrix for the best performing Hierarchical model with window size=20 showing high overall accuracy (0.75) and F-Score (0.74)**

| Class | Moving | Queuing | Stationary | Climbing |
|---|---|---|---|---|
| Moving | 1294 | 108 | 11 | 38 |
| Queuing | 130 | 414 | 85 | 1 |
| Stationary | 32 | 109 | 595 | 0 |
| Climbing | 259 | 44 | 6 | 203 |

TABLE III: **Class-wise Precision, Recall and F-score for the best performing Hierarchical model with window size=20**

| Metrics | Class Labels | | | | Average |
|---|---|---|---|---|---|
| | Moving | Queuing | Stationary | Climbing | |
| Recall | 0.89 | 0.66 | 0.81 | 0.40 | 0.75 |
| Precision | 0.75 | 0.61 | 0.85 | 0.84 | 0.76 |
| F-score | 0.82 | 0.63 | 0.83 | 0.54 | 0.74 |

### *B. Flat model*

Figure 8 illustrates the performance of various classifiers for a flat model. As the Figure shows, performance of all flat models is significantly lower than that of our hierarchical model. The best a flat model could achieve was an accuracy of only 0.58 which was with neural networks and a window size of 10.

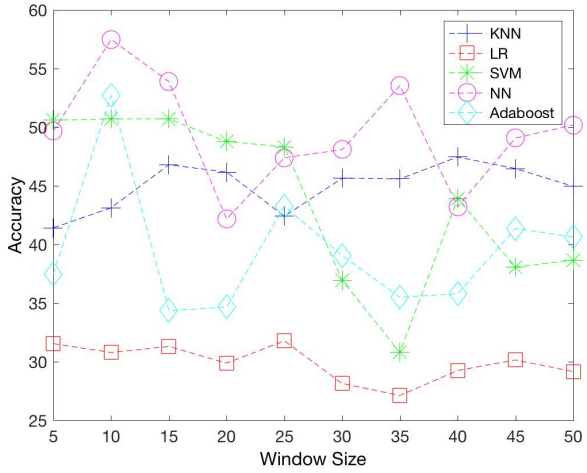


Fig. 8: Variation of Accuracy with Window Size (with KNN classifier, Regularized Logistic Regression, SVM, Multi-layer perceptron classifier and Adaboost) showing poor results for Flat Model

### *C. Key observations and interpretations*

Based on the results obtained, we can make the following conclusions:

- Beyond a window size of 20, the accuracy (and the F-measure) values decrease with the increase in window length. This is most likely due to the fact that as the length of the window increases, more number of windows span across readings of two different motion-activities (thus two different classes). This interferes with the learning algorithms resulting in unclear decision boundaries. It should be noted that the label of the first value of a window is selected as a training label for that entire window. These training labels are given by the user using the buttons in our Android application, as explained at the beginning of Section V.
- We observe that at level 1 of our hierarchy, all the classifiers perform equally well on our data. This is expected as we used only summarization (summarized to either 1 or 0) of the altitude values to represent a data-point (a window). As a result, the classes *Climbing* and *not-Climbing* become very easily separable.
- Our full hierarchical model, complete with classifiers with level 1, level 2 and level 3, performs best when we used SVM at level 2 and Multi-layer perceptron classifier at level 3. The ability of these two classifiers to classify data points which are non-linearly separated might be the reason of their better performance.
- Flat models do not perform well for our application domain.

## VI. Conclusion and Future Work

This paper proposed a hierarchical method for delineating the typical motion-related activities done at a metro station. The model focuses on *Queuing*, *Climbing*, *Moving*, and *Stationary* activities at the metro station. Such a model is useful in smart-city applications such as detecting crowds in the metro trains. Traditional CCTV based methods are not suitable for this problems due to challenges of data availability and privacy concerns. The proposed model uses accelerometer and barometer data from smartphones for delineating the four classes. We evaluated our models using real-data collected at Delhi Metro stations. Our results indicate that the proposed model outperforms alternatives by a significant margin.
We will exploit more classifiers to observe how they behave on this kind of data, as an extension to this work.

This paper was in part supported by the IIT Ropar, IIIT-Delhi and DST SERB (grant number ECR/2016/001053).

## References

[1] A. Anparthi, "Public cctv footage now a classified document," Jan 10, 2017.
[2] "Cctv footage can be preserved for seven days: Dmrc," Dec 28, 2012.
[3] M. Li, Z. Zhang, K. Huang, and T. Tan, "Estimating the number of people in crowded scenes by mid based foreground segmentation and head-shoulder detection," in *2008 19th International Conference on Pattern Recognition*, Dec 2008, pp. 1–4.
[4] A. Albiol, M. J. Silla, A. Albiol, and J. M. Mossi, "Video analysis using corner motion statistics," in *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, 2009, pp. 31–38.
[5] D. Conte, P. Foggia, G. Percannella, F. Tufano, and M. Vento, "Counting moving people in videos by salient points detection," in *2010 20th International Conference on Pattern Recognition*, Aug 2010, pp. 1743–1746.
[6] P. G. Kannan, S. P. Venkatagiri, M. C. Chan, A. L. Ananda, and L.-S. Peh, "Low cost crowd counting using audio tones," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '12. ACM, 2012, pp. 155–168.

[7] M. Vij, V. Naik, and V. M. V. Gunturi, "Use of ecdf-based features and ensemble of classifiers to accurately detect mobility activities of people using accelerometers," in *2017 9th International Conference on Communication Systems and Networks (COMSNETS)*, 2017, pp. 39–46.
[8] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson, "Cooperative transit tracking using smart-phones," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '10. ACM, 2010, pp. 85–98.
[9] H. Xia, Y. Qiao, J. Jian, and Y. Chang, "Using smart phone sensors to detect transportation modes," *Sensors*, vol. 14, no. 11, pp. 20 843–20 865, 2014.
[10] T. S. Prentow, H. Blunck, M. B. Kjærgaard, and A. Stisen, "Towards indoor transportation mode detection using mobile sensing," in *International Conference on Mobile Computing, Applications, and Services*. Springer, 2015, pp. 259–279.
[11] K. Sankaran, M. Zhu, X. F. Guo, A. L. Ananda, M. C. Chan, and L.-S. Peh, "Using mobile phone barometer for low-power transportation context detection," in *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '14. ACM, 2014, pp. 191–205.
[12] S. Zhang, P. McCullagh, C. Nugent, and H. Zheng, "Activity monitoring using a smart phone's accelerometer with hierarchical classification," in *2010 Sixth International Conference on Intelligent Environments*, 2010, pp. 158–163.
[13] Y.-S. Lee and S.-B. Cho, "Activity recognition using hierarchical hidden markov models on a smartphone with 3d accelerometer," *Hybrid Artificial Intelligent Systems*, pp. 460–467, 2011.
[14] X. Su, H. Tong, and P. Ji, "Activity recognition with smartphone sensors," *Tsinghua Science and Technology*, vol. 19, no. 3, pp. 235–249, 2014.
[15] E. Mitchell, D. Monaghan, and N. E. O'Connor, "Classification of sporting activities using smartphone accelerometers," *Sensors*, vol. 13, no. 4, pp. 5317–5337, 2013.
[16] S. L. Lau, I. König, K. David, B. Parandian, C. Carius-Düssel, and M. Schultz, "Supporting patient monitoring using activity recognition with a smartphone," in *Wireless communication systems (ISWCS), 2010 7th international symposium on*. IEEE, 2010, pp. 810–814.
[17] SensorManager. [Online]. Available: https://developer.android.com/reference/android/hardware/SensorManager.html