A PERVASIVE FRAMEWORK FOR IDENTIFYING ACTIVITY PATTERNS OF MOBILE USERS AND PREDICTING ACTIVITIES

Diyunugalge Chamika Sandun Weerasinghe

168275N

M.Sc. in Computer Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

March, 2020

A PERVASIVE FRAMEWORK FOR IDENTIFYING ACTIVITY PATTERNS OF MOBILE USERS AND PREDICTING ACTIVITIES

Diyunugalge Chamika Sandun Weerasinghe

168275N

This report is submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science specializing in Mobile Computing

Department of Computer Science and Engineering

University of Moratuwa

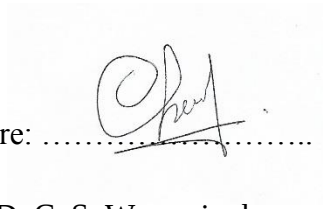
Sri Lanka

March, 2020

DECLARATION

I declare that this is my own work and this report does not incorporate without acknowledgement any material previously submitted for the degree or diploma in any other university or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles of books).



Signature: …….. Date: 2020/03/15

Name: D. C. S. Weerasinghe

I certify that the declaration above by the candidate is true to the best of my knowledge and that this report is acceptable for evaluation for the CS-5999 PG Diploma Project.

Name of the supervisor: Dr. Indika Perera

Signature of the supervisor: …….. Date: ……

ABSTRACT

Smartphones has become one of the most used devices in day to day life. Even though they already have so many features, they still lack the ability to identify user's context and the intentions. This is important for improving user experience and make existing mobile application more user friendly. The issue is that there is no underlying support either from operating system or software level to predict the user's intensions based on user context.

The main objective of this research is to come up with a framework to predict user intentions based on user context by identifying activity patterns. The framework must be run in-device so that it will function irrespective of the network connectivity.

We selected "clustering" as the approach because it does not involve high computation power or complexities to run in-device. We identify activity patterns by clustering the user's actions and then predict based on the closest cluster for the given time. We have evaluated K-means and Expectation-maximization (EM) clustering algorithms for compatibility for the framework. Unlike computers, mobile devices do not have powerful CPUs or memory. Therefore, we measured CPU time and memory usage of these algorithms to select the best. To maintain low-end device compatibility, we tuned in the algorithm parameters to achieve high accuracy keeping the CPU and memory consumption in low levels.

In conclusion, we have successfully identified that EM clustering is suitable for high-end devices and it gives high accuracy while K-means is suitable for low-end devices with acceptable accuracy. We have implemented the framework as an Android library and developed a proof of concept application by embedding the implemented library to show that this research will actually enables application developers to give better user experience to their applications.

ACKNOWLEDGMENT

I owe my deepest gratitude to my supervisor, Dr. Indika Perera, for his invaluable support in providing relevant knowledge, advice and supervision throughout the project. This would not have been possible without his expertise and continuous guidance.

Finally, I would like to thank my colleagues at CodeGen Int. Pvt Ltd and Bhasha Lanka Pvt. Ltd for covering my work and helping me to balance the workload. Without them, this project would not have been possible.

Last but not least, I am grateful for all the people who supported me throughout this research in various means.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Description
IoT	Internet of Things
HMM	Hidden Markov Model
MSE	Mean Square of classification Error
NN	Neural Networks
KNN	K-Nearest Neighbors
ECA	Event-Condition-Action
GPS	Global Positioning System
API	Application Programming Interface
SDK	Software Development Kit
IDE	Integrated Development Environment
App	Mobile Application
EM	Expectation-Maximization

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

1.1 Background

People use a number of digital devices to interact with the world around them in their day to day activities. From turning off the alarm in the mobile phone in the morning to turning off the bedroom light at night, these interactions follow a pattern specific to a user with a sequence of repetitive actions. Let's take a person who does a 9 to 5 day job for an example. They wake up at specific time, commute to work, leave work around same time, commute home, eat dinner and sleep. People use a number of devices to get help with these activities including mobile phones and virtual assistance devices like Alexa and Google Home. These devices can be set up to interact with each other as well.

The concept of "IoT (Internet of Things)" is based on the idea of machines and the devices communicating with each other. It will be helpful for the machines to learn the user's behavior to make the best use of the machine to machine communication.

Currently, more people in the world have access to a mobile phone worldwide than clean running water [16]. In some cases, people tend to use more than one mobile phone. According to the office of national statistics in UK, in 2018, among all adults, 78% used mobile phones or smartphones to access Internet [17].

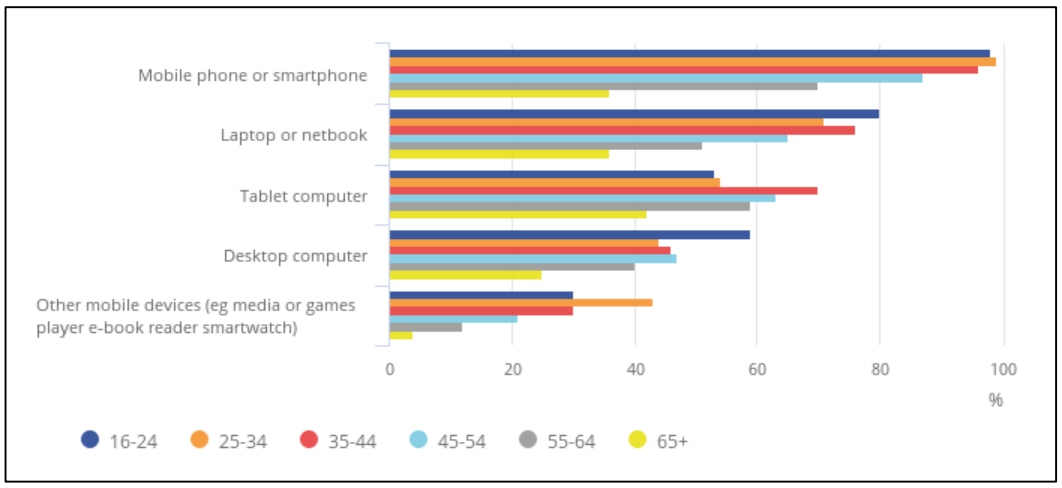


Figure 1.1 Devices used to access the internet, by age group in UK 2018

Mobile phones have various models and makes with a number of different sensors embedded. The count of the sensors bundled in high end smart phones, sometime goes up-to more than 10. But majority of the smart phone users do not use these sensors. The number of mobile applications which uses these sensors has always been low. For example, GPS sensor is only used with applications dedicated to navigation like Google Maps and the light sensor is used only for detecting environmental light intensity based on which the brightness of the screen is regulated.

The opportunity to use these sensors and develop something beyond their typical use cases is always there. One good example for this is using the mobile phone camera and flashlight to detect the heart rate of the user. When the user put the finger in both flashlight and the camera, the finger will be light up by the flashlight in red color. The color variation due to the blood flow in the finger can be identified by the camera [18].

We can use these sensors to capture the data frequently and identify the user activities or the user context. Additionally, data recorded from the mobile applications such as call records, SMS messages can also be used for the same functionality. Once the activities are mapped with user context it can search for identifying patterns. These patterns can be used to predict future user activities depending on the user context for a given timeframe.

1.2 Problem Statement

Smartphones has been improved a lot in the space of sensors and applications. But people still have to manually do the same thing over and over again. Consider a situation where user drives to work every day and open Maps while driving to find the best route. With the amount of sensors mounted in the mobile devices, we can identify this behavior pattern and open the Maps application or suggest opening the maps application when the user gets into the car in the morning.

The main research problem we are trying to address from this research is to find a mechanism to reduce these repetitive actions by users by leveraging the data captured from the embedded sensors and the installed applications in the mobile.

1.3 Objectives

The main objective of this research is to develop a pervasive framework to identify the user behavior patterns and predict the activities in real-time, so the user does not have to do repetitive actions.

1.3.1 Other Objectives

- Research on how to identify the behavior patterns from the user activities and prediction.
- Come up with a good solution that suites both high-end and low-end devices without significantly draining battery.
- Develop a framework for a selected mobile operating system to predict the user actions based on the behavior patterns.
- Develop a proof of concept application by using the developed framework.

1.4 Overview of the Document

This document consists of five chapters. The first chapter gives the introduction to the research by presenting the background of mobile devices and sensors. It will present the research problem which we are trying to solve and the objectives of the research.

The second chapter contains the finding of the related literature. Starting from the context identification mechanisms which includes data collection and feature extraction. It is followed by activity recognition and activity prediction.

The third chapter describes the identified methodology to solve this problem. This includes data collection, data training and prediction, evaluations of the selected algorithms and the implementing the framework.

Fourth chapter contains the information regarding the high-level flow of the framework, system architecture and implementation details of the data collection and proof of concept applications.

In the fifth chapter we have included the results of different approaches we took when predicting and their accuracy levels.

Last chapter was dedicated to discuss the research contribution from the research along with the limitations of the approach we took and future directions where we can extend the framework.

CHAPTER 2

LITERATURE REVIEW

2. LITERATURE REVIEW

This chapter is organized in the following way.

- Context identification by collecting the sensor data and feature extraction
- Activity Recognition
- Activity Prediction

2.1 Context Identification

User context is a broad topic and it can be what a user is doing, where the user is living, how the environmental and user condition etc. Identifying the user context is challenging for the computing devices due to the variations of different user behaviors. Several solutions are proposed in past research work to identify context using sensors but accuracy need to be improved on these solutions. A generic process to identify the context is visualized in Figure 2.1 [1].

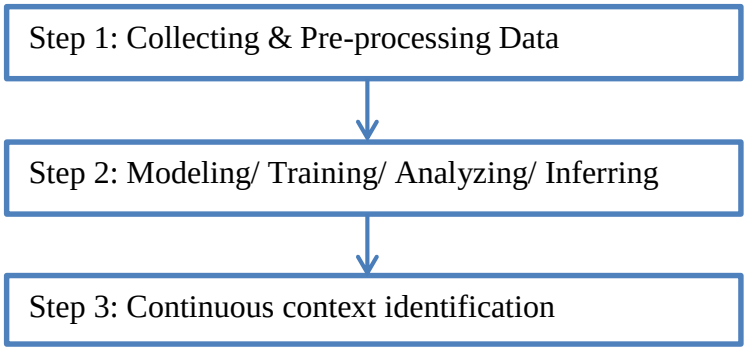


Figure 2.1 Generic Process for identifying context continuously

In the Step one, raw data is gathered from the multiple sensors and stored for later processing. Pre-processing is done in order to improve the efficiency for the later steps. For example, data is gathered as a stream and stored as a batch of data sets. This preprocessing step will reduce the number of disk operations which eventually affects the energy consumption of the mobile device as well as make the raw data more organized so that it will simplify the processing in later steps.

In step two, high level information is generated by processing the raw sensor data. This is done by modelling, training, analyzing and inferring. A suitable method is

selected based on the sensor data and the information retrieval mechanism. For example, if you need to identify whether the user is walking, you can use accelerometer data, model it, train them to identify the walking pattern of the user and analyze to generate the required information.

User's context is continuously changing. Identifying this context is discussed in the third step. When the context keeps changing, the process needs to adapt its results based on the changing sensor readings. These identified high-level context will be served for other applications in this step.

Context can be categorized into 3 types.

1. Physical Activity – User's actions such as sleeping, sitting, walking, running.
2. Social Interaction – User's interaction with other people.
3. Environmental Interaction - User's interaction with the environment like movement pattern.

2.1.1 **Data collection using mobile device sensors**

Smartphones have several built-in sensors to accommodate services like navigation, brightness control, automatic screen orientation detection, device health detection etc. A list of sensors which comes with average level smartphone and their usages are given below. [2, 3]

1. Accelerometer – Identify device orientation and change screen orientation based on that.
2. Gyroscope – Works along with accelerometer to enhance the accuracy of position level and orientation of the device. This uses the principle of angular momentum which makes the sensor readings independent of the gravity.
3. Digital Compass – Uses in navigation applications to identify the direction of the phone.
4. Global Positioning System (GPS) – Finds the GPS location of the device, altitude and location related information

5. Proximity sensor – The main application of the proximity sensor is to turn off the screen when a user put the phone in the ear while a call is ongoing. It also uses to identify whether the phone is the pocket or covered by a device case.
6. Light sensor – Captures the light intensity and controls the screen backlight brightness based on the reading.
7. Temperature sensor – Uses to detect the internal temperature of the device. If the internal temperature is high, system informs or act to switch off the device as a safety precaution of the internal electronics.
8. Microphone – Captures the voice of the user in a call. Some devices have a separate microphone to capture environment noise. Then environment noise stream is used for noise cancellation for better output sound quality. Also, the input voice will be filtered by using the environment noise.
9. Camera – Captures and generate digital image. Some devices contain dual cameras to detect depth of the objects to correctly identify the depth of field.
10. Global System for Mobile Communications (GSM)/3G/4G module – Makes voice calls or initiate data connection with networks.
11. Infra-Red (IR) sensor – Enables the communication using Infra-red.

Other than the above sensors some high-end devices have pressure, humidity, fingerprint sensors. Considering the number of sensors embedded in a single device makes the device ideal as a source of raw data for the identification of user context.

2.1.1.1 Challenges in data collection using mobile device sensors

Using all these sensors do have some challenging aspects when used frequently. Following are some of the challenges which has significant impact.

1. Energy [4] – Although the mobile devices embed a variety of sensors, they are powered by a battery. Because of the design constraint which is to minimize the size of the mobile device, battery dimensions must be reduced. This limits the capacity of the battery. When using many sensors, they need extra energy to function which consume the mobile device battery. Therefore, enabling sensors to capture reading continuously throughout the day is nearly impossible. If it does an average mobile device will only last up to 4 or 5 hours.

2. Heterogeneity – Different device manufactures use different electronic modules to produce sensors. This differentiate the sensor accuracy and range. Also, operational environment will change depending on the user [22]. Therefore, implementing a single solution with high accuracy for multiple devices and sensors is challenging. The behavior of the users also varies from each other. Walking pattern of a user might be different from other due to the height, weight and other factors of the human body. This makes the problem more complex because the same device may have different sensor readings for different users.

2.1.2 Data collection using mobile device applications

Most of the research work focus on identifying context via sensors. But there is another aspect when it comes to mobile phones. Users tend to install many software/applications and interact with them throughout the day. This can also be an opportunity to learn the lifestyle of the user. Many social media applications like Facebook, Twitter and communication applications like WhatsApp, Viber will be a good source for data as well. Also, call logs and SMS messages are two main sources to import data as many mobile operating systems are allowed to use these data with the consent of the user [23].

Not all applications are data or event producers. Some applications do not even support data exports. This is the main limitation of using mobile applications to collect data. If the application and/or operating system supports data collection from applications, this data can also be used similar to sensor readings. A good example would be sending an SMS message to your partner when you reach office in the morning every day. A pattern should be able to identify from this SMS sending event.

2.1.3 Preprocessing

The output data from the sensors, can be captured continuously and recorded in the device itself. This raw data should be refined before it is used for context recognition. A sensor calibration is a better way to increase accuracy of sensor readings. Although the sensors are heterogeneous it can achieve a good sensor

measurement, if the readings are calibrated for the sensors in a standard way [3]. This should be done by the users before starting to record the readings. A simple mobile devices activity can be used to calibrate each sensor. As an example, informing user to rotate the device horizontally can calibrate the accelerometer readings in a single axis. When the sensor measurements are occurring, sensor readings can be modified according to the calibrated values.

Apart from sensor calibration, sensor readings may have redundancies and noise. The preprocessing step should be able to remove them as well. But the preprocessing must not exceed the high computation and memory requirements. Heuristics can be used for preprocessing of raw data. These modified, "cleaned" data can then be transferred for further processing.

2.1.4 **Feature Extraction**

Having raw data and process them to identify activities is impractical. Because of that we need feature extraction. Feature extraction is the process of distilling the raw sensor data and converting it into more computationally efficient and lower dimensional forms called features [3].

Whether the data comes as a stream or sequence of events, it is necessary to split the data into samples. Splitting window should be depended on the nature of the sensor readings, resolution and sensor type. For an example having small (ex: 2 seconds) window for accelerometer reading is acceptable since the changes of the readings are rapid. But having small window for location readings will not be suitable because the location changes in small intervals are negligible.

Generating accurate features out of raw data is important for context recognition. Feature extraction in three types of context can be described as follows.

2.1.4.1 Feature extraction in physical activity

Since the mobile phone has become a ubiquitous device for sensing, physical activity of a user throughout the day can be accurately identified. Accelerometer, gyroscope, proximity and GPS sensors can be used to collect user's physical changes [5,6]. This can be helped to identify the duration of activities like sleeping, sitting, walking,

running, travelling etc. [25]. Mean, standard deviation, number of maxima in locality are some of the features which can be extracted based on the time. Google awareness library is a good production ready Android library which has implemented identifying activities like on foot, walking, running, on bicycle, in vehicle. Although the internal functionality of this library is not published, we can identify that it uses 3-axis accelerometer, gyroscope, compass and GPS sensor readings to recognize user activity.

2.1.4.2 Feature extraction in social interactions

Social interaction covers the context of human interactions. For this microphones, proximity sensors and shared locations can be used. Basically, it need to know whether the person is communicating or being among the others. Time based feature extraction method would be analyzing the intensity of the sound against the time.

2.1.4.3 Feature extraction from environmental sensing

Location based readings such as GPS coordinates, places and altitude can be captured from the mobile device sensors. These reading can be used as an input for retrieving weather data to capture the environmental context. Apart from that sound, light, temperature, and humidity are some of the measurements which can be measured under environmental sensing. Time based averaging, standard deviation are basic level feature extraction techniques for this. Also probabilistic analysis methods can be used since are more likely to have patterns.

2.2 Activity Recognition

Context inference is the way of identifying activity of the user. Here the activity is considered as various user context as well as the events generated by users like sending a SMS message, making a call, updating or publishing posts in social media. The features which are extracted from raw data is the input of context inference algorithms. These algorithms are also known as classification algorithms.

Classification algorithms need initial data set to learn the behavior of data. This is called the learning phase. The algorithm tries to identify the patterns in each dimension of expected features. There are several types of algorithms which can be

used for activity recognition such as threshold-based algorithms, neural networks and hidden Markov models.

2.2.1 **Activity recognition through learning**

Learning in activity recognition can be categorized into two models [24]

1. Supervised learning model
2. Unsupervised learning model

Unsupervised learning models does not use any kind of trained data.

2.2.1.1 Supervised learning model

Supervised learning model is the process of learning through already defined algorithms and the data is provided along with the classes. Accuracy of the recognition is dependent on the trained data [24].

Supervised learning is the Data mining task of inferring a function from labeled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way

There can be deviations between the trained data and actual data based on the

There can be deviations happening because the trained data is based on general patterns. If the data set is significantly deviates from the trained data, generalization error will be high. As a measurement of the error mean square of classification error (MSE) can be presented [3].

$$MSE = Noise^2 + Bias\big(f(x)\big)^2 + Var\big(f(x)\big)$$

$x \quad = input\ vector$

$f(x) = estimation\ of\ the\ classification\ model\ for\ the\ class\ x$

Noise is the error of the system or sensors. This is caused by the system or sensor errors. *Bias* is the error causes by the learning method. *Var* is the variance of the error related to classification specific sensitivity.

Noise cannot be reduced. In order to minimize MSE is to minimize the other two factors. If the training data set is large, it may cause high bias error (under-fitting). It may cause high variance error (over-fitting) when using small training data set. Therefore, in both type of classifiers may face increased MSE. To overcome this issue, a data set collected from multiple users can be used. Because the actual data from users will be less deviated from the real users. This data can be used as training data set to minimize MSE. The problem arises when using multiple users for training data sets is to estimate number of users for better error reduction.

The users are the best resource for evaluating classified results. There are three main techniques to get user's contribution for evaluation results [3].

1. Active learning
2. Community Guided learning [26]
3. Hybrid approach of above [27]

Active learning interacts with user to evaluate the classified results. Initially a test data set is used to gather some results. Then it asks the user to evaluate the correctness of the classified results. Based on that it will continuously changes the classification.

In community guided learning uses crowd sourcing as a generalizing method. Unlike single user based technique in active learning, erroneous evaluations of the users can be omitted and generalization accuracy can be increased. But it will take considerable amount of users to contribute for a better accuracy.

Hybrid approach of active learning and community guided learning was suggested to improve classification further. A service based recognition is used for active learning and the learnt data is transferred to a backend server for accumulation. Community guided learning technique is used in backend servers where multiple user classifications are received. Then personalized information can be generated based on user's preferences.

2.2.2 **Different classification techniques**

The selection of learning technique is depended on the selection of the classification technique. Selection of classification technique should be decided considering the time and space complexity of the algorithm. Since the target is to process within the device, algorithms which demands high computing power is not suitable for the application. This document focuses only the algorithms which are relatively simple and can fit to the mobile device computation.

There are two types of classification algorithms based on the optimization approach [28].

1. Generative algorithms
2. Discriminative algorithms

Generative algorithms are trying to optimize in the assumption of existence of probabilistic relationship between the data and the classes. Also it specifies that there exists a joint distribution between the features and the classes. Mean posteriori, maximum posteriori and maximum likelihood are some examples for generative algorithms. Generative models are based on probability computation and most of the time they are not using because of the computational costs.

Discriminative algorithms are using different approach introducing the distance or similarity between the any pair of patterns. Simply it need to define a distance or similarity between samples. If it is in the same class, the similarity will be higher (distance is lower) and vice versa.

The following table summarize the approaches of generative algorithms and discriminative algorithms.

Table 2.1 Classification Techniques

Generative Algorithms	Discriminative Algorithms
Hidden Markov model	Neural networks
Bayesian networks	Decision trees
Discriminant Analysis	Hierarchical thresholds
	Fuzzy logic
	Clustering

2.2.2.1 Generative Algorithms

2.2.2.1.1 Hidden Markov Model (HMM)

Hidden Markov model has a chain of finite states set. The states are not observable and it is said that they are hidden. State transitions associates with a set of probabilities and states is associated with a probability distribution. The Markov model can be two type of patterns. Left-to-right model and the ergodic model [3].

(a) (b)

Figure 2.2 (a) left to right HMM model (b) ergodic HMM model

In left to right HMM model the transition happen only in one direction. In contrast, ergodic HMM can happen all possible transitions which may have cycles, multi-directions.

Using HMM researches have come across several solutions which related to context awareness and prediction such as characterizing activity pattern[7], identifying user

mobility and prediction[8], situation identification of mobile user[9], sequential behavior prediction[10].

2.2.2.1.2 Bayesian Classifier

From the Bayesian classifier following equation can be applied for the sensor data classification as a component of probabilities

$$p(context|sensor\ data) = \frac{p(sensor\ data\ |context) * p(context)}{p(sensor\ data)}$$

As the equation depicts, the computation of Bayesian classifiers are easy since it only having the productions of probabilities. Therefore it is even suitable for processing within a low end mobile device [11]. Even context identification in mini wearable devices are done using this [12]. Using Bayesian classifiers should be carefully selected depending on the data set [13]. Let's say the selection is Naïve Bayes, then it would be more accurate to use data points which are locally independent than the dependable.

2.2.2.2 Discriminative algorithms

2.2.2.2.1 Decision Trees

Decision trees is a mathematical tree consisting several nodes and edges. It will generate inner nodes, branches, and leaves. The simplest decision tree would have two branching factor where each node evaluates the attribute and output true or false (yes or no). Following figure describes the decision tree of two branching factor.

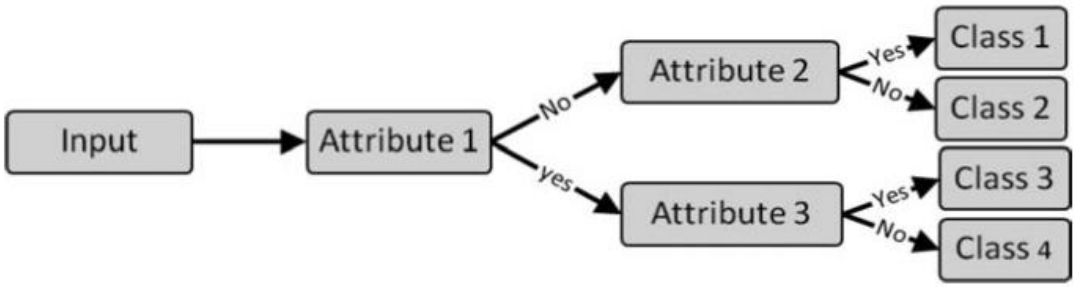


Figure 2.3 Sample decision tree having branching factor of 2

When the data set is entered to the input of the decision tree, each node evaluates the data. Depending on the evaluation next node will be decided. Then the next node is do the same thing and the process continuous until a leaf is found in the decision tree. The leaf of a decision tree is a class according to that data. Then the classification can be done. The evaluation of data via a decision tree is simple and the complexity is depended only on the height of the tree along a path. Therefore, decision trees are considered as computationally efficient. There are practical implementations of decision trees to recognize user activities such as walking, running, sitting and standing [3].

But the creating the tree structure is complex due to consideration of several aspects such as training and learning take much time. If the target is to build the decision tree within the device is time and processor consuming. Therefore, prebuilding the decision tree in a generalized way is a solution. But the variance of different users, makes this solution produce less accurate classification although the computation is low in the mobile device.

2.2.2.2.2 Neural Networks (NN)

Artificial Neural Networks are created and used to classify data. Neural Networks are complex, need parallel computation and have non-linear model [14]. Same as the HMM, neural networks have two types. Feed-forward model and feedback model. Feed-forward works in one direction and it does not allow data to go through the same node twice. In contrast feedback model enables data to iteratively evaluate by traversing within the network. It allows output of a node which can be processed data, to be used as an input of another node.

The states of the neural networks are hidden. The structure of a NN is shown below.

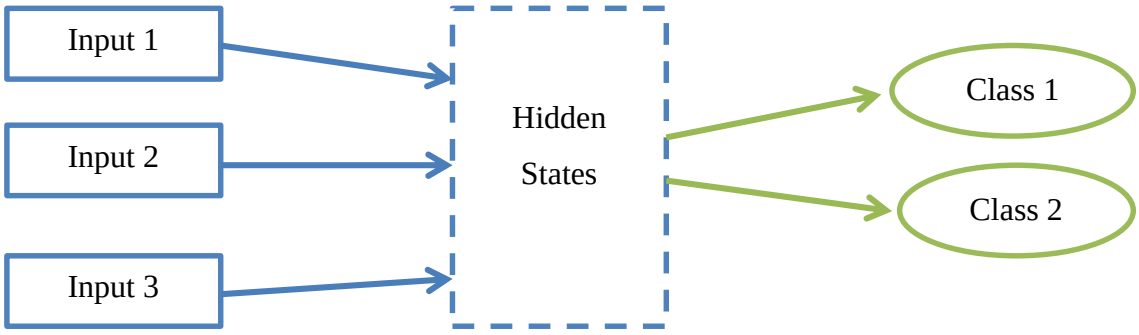


Figure 2.4 Structure of a general Neural Network

Neural Networks are computationally expensive because creating neural network and evaluate through a neural network come across many operations. If the NN allows feedback loops, the number of operations will be even higher. Same as the decision trees training NN for a data sets are expensive. Therefore, building multiples of less complex NN which focused on single task is better than using single NN which focused on multiple tasks. Because large NN consume higher degree of both space and computation. But if the NN is simplified by focusing on single task like identifying walking pattern, the space and computation complexity will be less and it will fit the low-end mobile device hardware. If the number of classification is higher, this cannot be scaled because different NN is needed for each classification.

2.2.2.2.3 Hierarchical Models

Hierarchical models define a set of thresholds for different classifications. The remaining process is somewhat similar to decisions trees. In decision trees the evaluation in the node is based on the training data set. But hierarchical model use thresholds which already defined from the start. This removes the complexity caused in creating the decision tree. Hence hierarchical model is considered as computationally efficient algorithm in classification. [11,12].

Defining thresholds is done by empirical experiments. As an example, separate thresholds along an axis will be defined for both running and walking by experiments. So that the model defined general values for the activities. There might be situations where these values are out of the thresholds due to the variations of users. This over-fitting issue cannot be prevented. Therefore hierarchical models must be used in the situations where generalize thresholds are within the range of almost all the possible users. The drawback of this method is the thresholds are predefined and they are depend on the supervised training data set.

2.2.2.2.4 Fuzzy Logic

Fuzzy logics does not have hard wired decision taking mechanism and this is similar to human decision making. It is able to classify even the data is inaccurate and partial. Fuzzy logics support approximate reasoning which helps in the applications

where a distinct activity cannot be identified, but the data suggest closer to some activity pattern.

Fuzzy logic has a knowledge base having set of rules. When an input comes, it evaluates the input against these rules. While evaluating the rules, it calculates a score using the assigned membership value or fuzzy truth. Finally, the output having the maximum fuzzy truth or score will be considered as the result. By comparing fuzzy truth with the other values, reasoning can be identified.

2.2.2.2.5 Clustering

All the above classification techniques are needed a training data set to initialize the model. Most of these supervised learning algorithms have similar complexity where the model creation using training data set is high and the evaluation is low. But clustering is different from those algorithms because it does not need a supervised data set. Therefore, it is categorized as an unsupervised learning algorithm for both classifications of patterns and calibration. After the clustering is finished, separate clusters suggest the similarity of elements. Therefore, a single cluster can be used as a class.

The main idea of clustering is based on the distance between the elements. There should be a function to calculate distance between two elements. If the distance is low, they are assigned to be in the same cluster. The area of a cluster can be defined through the maximum distance between the farthest two elements.

2.2.2.2.5.1 K Nearest Neighbor Clustering

KNN algorithms use clustering approach to classify data. Here the distance between the elements is considers as the nearest neighbor distance. The proximity of each other is calculated and this distance value is used for clustering. KNN nearest neighbor distance support multidimensional feature space. Therefore, multiple types of features can be accounted when the proximity is calculated.

When applying KNN algorithms for classification, it initializes with a trained data set which forms a set of clusters. These are the labeled data. Then the input data is inserted and based on the multidimensional proximity newly inserted data is placed.

New data is labeled based on the distance from the new data to the labeled data. In most cases, cluster of closest labeled element is considers as the label of the new data. Then it can classify and also a mathematical value can be calculated for representing the reasoning for the classification.

Researches were done for identifying activity by using this method [14]. They used Euclidian distance and trained data was based on mobile position specific to the user. The targeted activities were predefined by the users. They could achieve 70% accuracy for all the defined activities.

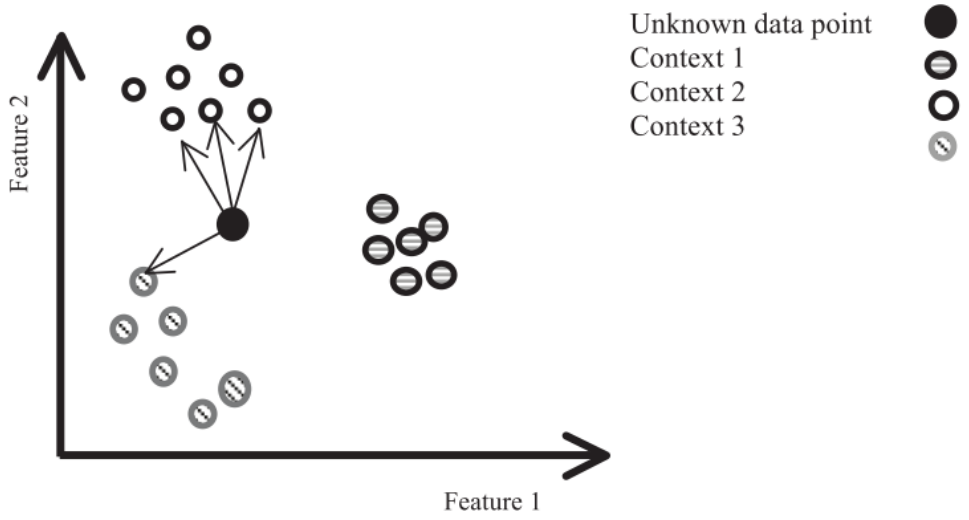


Figure 2.5 Placing the new data point in K-Nearest Neighbor Clustering [3]

2.2.2.2.5.2 K-Means Clustering

In K-means algorithm a multidimensional space of features is divided into K cluster. When forming clusters, it tries to find the optimum position of the cluster centroids. A recursive algorithm is used when finding the optimum position. This algorithm is computationally efficient but the data need to be saved in memory. Therefore, memory consumption is higher than other algorithms. This limits the usages of this algorithm in low end mobile devices if the dataset is large. As a result of this instead of directly using the algorithm, it uses to calibrate the data and then use another algorithm for classification. Some researches like physical activity detection by

decision trees [13], achieved successful results by using K-Means clustering for smoothing the classification results.

2.2.2.2.5.3 EM Clustering

Expectation-Maximization clustering is an advanced version of K-means clustering. In each iteration of EM clustering the probability of owning a certain data point to a cluster is calculated. Then the cluster's mean and variance changed with the updated ownership. Therefore, each iteration, the clusters may change their mean and variance hence their distance to the data points [29]. Unlike K-means clustering, there is no distinctive assignment of a data point to a single cluster. EM clustering will have only probabilities to each data point. Basically K-means is a subset of EM clustering where probability is 0 or 1.

2.3 Activity Prediction

Once the user context and activities are identified, it can identify the patterns of activities. Most of the users have repetitive behavior in daily tasks. Therefore, it is easy to identify patterns since the overfitting problem will be reduced if the recorded data set has a considerable time span. Predicting activity is the idea of knowing the user's intentions. As mentioned earlier, if the user has repetitive life style, it tends to follow sequential list of events. These events are depended to the previous event. As an example, once the user enters to the home, he turns on the WiFi in the mobile device. Here turning on the WiFi is dependent on the event entering the home.

One of the successive activity prediction techniques is the ECA model.

2.3.1 Event-Condition-Action (ECA) Model

ECA model has event driven architecture and define a set of event and action relationships. This relationship has 3 attributes which are Event, Condition and Action. After recognizing context, this set of relationships are identified by the algorithms which are used in context classifications. The application of the algorithm is different but the same approach can be taken for identifying patterns in the user context. Once the patterns are recognized, relationships will be revealed. These

relationships are converted into set of rules. In natural language a rule has the syntax of following manner.

$$\text{On } < \text{event} > \text{ If } < \text{condition} > \text{ Do } < \text{action} >$$

Considering above example, the rule would be

$$On\ < reaching\ home > If\ < user\ location\ is\ home > Do < WiFi\ On >$$

ECA rule managers, ECA rule engine are necessary components of evaluating rules. ECA rule manager generates the rules based on the user activities and actions. Both actions and activities are found within the user context. For an example, walking is an activity. But it will be an action if the user always walks at a specific time. So the user activities can both be events as well as action.

A proposed system architecture [2] of a related research is shown in Figure 2.6 ECA architecture for activity prediction.

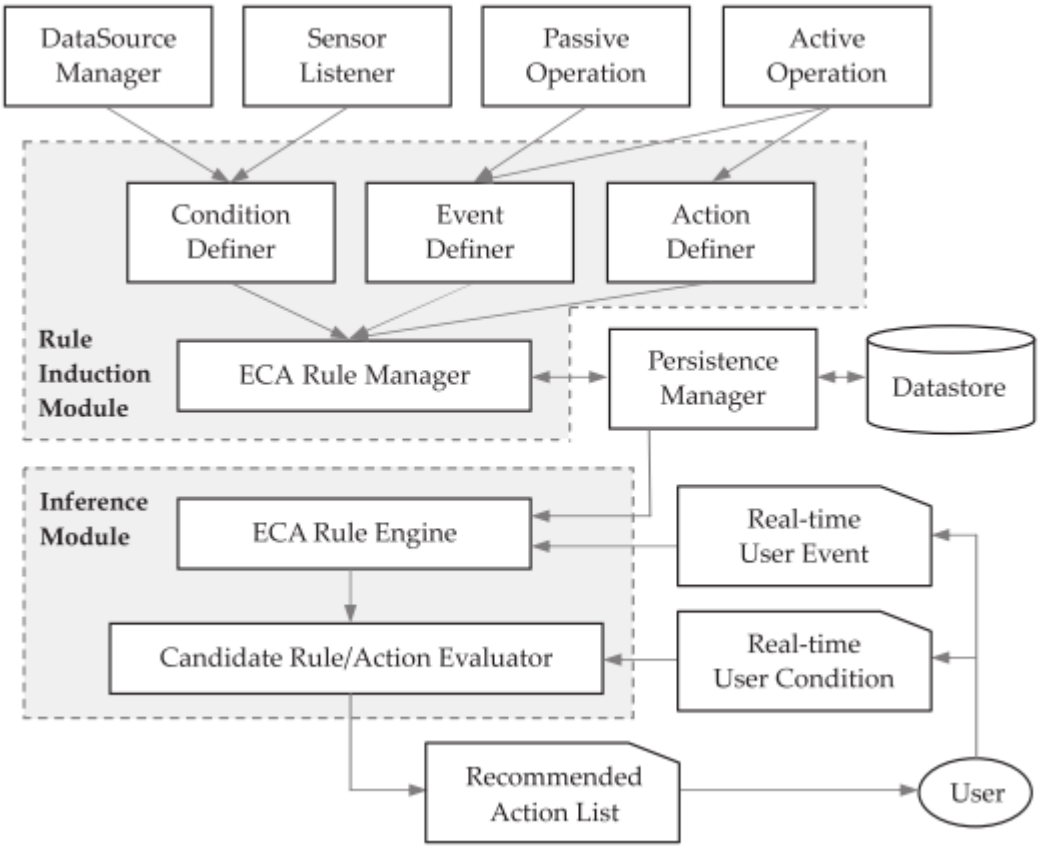


Figure 2.6 ECA architecture for activity prediction

Rules which are generated by ECA Rule Manager are consumed by ECA Rule Engine. If the context changed, the change is forwarded to Rule Engine and it will evaluate and select one or more rules to trigger. Action of the rule is triggered automatically or give a list of actions to be selected by the user.

2.3.2 **Clustering**

The same clustering algorithms discussed in section 2.2.2.2.5 can be used for pattern recognition as well. Because one the user context and the previous actions are collected, they can be fed into a clustering algorithm to generate clusters. If the data point dimension includes temporal data, the generated clusters will represent a collection of similar events. Theoretically a collection of similar events is a repetitive actions/activities. This special behavior of clustering can be applicable to this research as well.

2.3.3 **Analysis**

The advantage of ECA model over clustering is its low complexity when predicting activities. Given a state of the user context, ECA model can iterate through the rules and if the conditions match it can generate the predictions. However, generating rules for ECA model is complex compared to clustering. Hence ECA model might not be suitable for low-end devices. The advantage of clustering is the simplicity of the model to generate predictions. Once we find the closest cluster, we can directly generate predictions using the data in the closest cluster. It does not involve complex steps like rule generation in ECA. Therefore, we can see that clustering is more suitable for low-end devices.

CHAPTER 3

METHODOLOGY

3. METHODOLOGY

3.1 Proposed Solution

The proposed solution is to implement a mobile framework (library) that mobile application developers can be used to retrieve list of predictions based on the context of the mobile user at a given time.

3.2 Workflow

The most important decision of this research is the selection of prediction method. We have to consider whether the method is compatible to run on a low end device. Since mobile devices has variety of hardware specifications, our framework should be compatible at least 80% of the devices available in the market. In order to achieve that, we need a prediction method which needs low CPU time and consumes low memory. Out of different prediction methods discussed in the literature review, we decided to go with clustering as it does not need high CPU time or large memory to run within the device compared to other prediction methods. It may take considerable CPU time when running clustering algorithm but by comparing accuracy, CPU time, memory allocation of different clustering algorithms, we can select a suitable clustering algorithm which fits low end devices. For this research we decided to evaluate K-means clustering and EM clustering algorithms. Generally, EM clustering consumes higher CPU time than K-means clustering, but it has higher accuracy and it is worth evaluating before making the final selection. Although algorithms like hierarchical clustering has low memory consumption and low CPU time, its accuracy is low compared to K-means [20].

The process of this research can be broken down to following main steps

1. Data collection
2. Clustering data and activity predicting
3. Evaluation
4. Implementing the framework

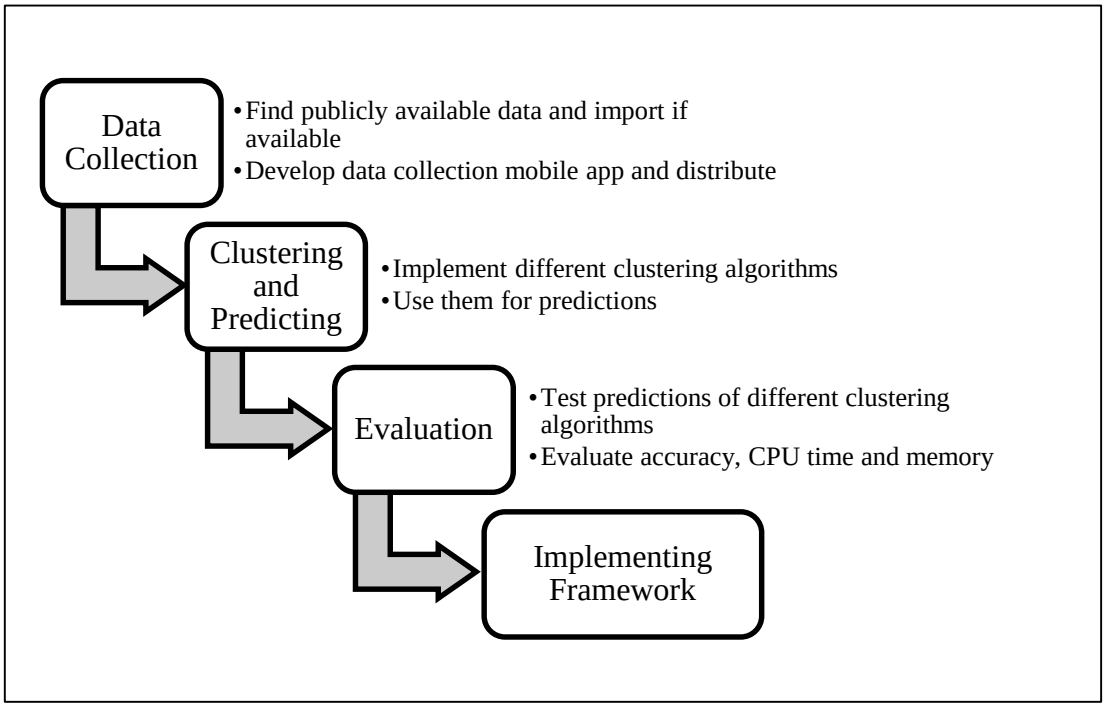


Figure 3.1 Research Workflow

3.2.1 **Data Collection**

There are two types of data sources when collecting data. First type is sensor readings and their processed outputs such as GPS location, user environment temperature, device battery level and light intensity. There can be processed output of sensor readings such as walking, running identified from accelerometer data. The other type of data source is software such as installed mobile applications and operating system. Call logs, sent and received SMS messages, opened applications and their statistics, interactions in social media applications, connected Wi-Fi networks are some of the data which are included in this type.

Both types of data sources generate data continuously irrespective of the device is being used or not. Therefore, instead of recording everything, we needed to decide what data should be recorded. Then the selected data is generalized to be recorded in a common way. We call these generalized data as "Events". An event must contain following fields.

- Action type – Whether the event is automatically triggered or caused by user interaction
- Event type – The type of the event like SMS message, call, user activity
- Timestamp –The date and the time of the event
- Class – The main action of the event like which phone number the user called or which mobile application opened.
- Event data – There can be several data bound with an event. These multiple parameters are recorded with data

For this research we decided to consider only the following common types of events

1. Call records – Received and initiated calls by the user.
2. SMS records – Received and sent SMS messages.
3. Activity – What activity the user is doing at a given time. Walking, running, staying are some of the activities.
4. App openings – Opened mobile applications by the user.

Although the user's exact location can be captured, we decided not to collect that because it raises higher level of privacy concerns. After deciding what type of data we need to collect, we searched for publicly available data repositories which has above types of events. Unfortunately, there were no publicly available data which we can be used for this research. Therefore, we had to develop a separate mobile application to collect data from different users.

3.2.1.1 Data collection mobile application

The data collection mobile application was developed to capture SMS, Call, mobile application usage and activity event. We gave the application users the option to select which data need to be captured. If the user is uncomfortable with sharing data with us, he/she could opt-out certain event types. For an example the user can opt-out sharing SMS events while enabling call and application usage events. The app was developed to upload persisted events to a cloud location at a configured time of the day. The data collection and upload tasks were designed to run in background so that user did not need to trigger anything after enabling data capture.

A main concern when collecting data is the privacy concerns. The users do not like to share the information of calls and SMS messages. The SMS message body is not collected by the application so that it will not be an issue. But exposing contact number was a privacy concern. Therefore, we used a hashing method for all the contact numbers. A given phone number had the same hashed text, which helped us to distinctly identify the similar SMS and call events while ensuring user's private data.

We distributed this mobile app among ordinary mobile users and asked them to select the event types they like to share and start the application. Also, we informed them that we need data for at least 30 days so that we can collect considerable amount of events for each user.

3.2.2 **Clustering Data and Activity Predicting**

In this stage, we try to implement different clustering algorithms and feed collected data into each clustering algorithm. Once the clusters are formed, we use them to generate predictions. The generated predictions are then evaluated later.

3.2.2.1 Preprocessing

The outcome of the data collection contains a list of different types of events occurred in different timestamps. The collected data cannot be directly feed into a clustering algorithm because collected data do not have different attributes. It only has timestamp of the event as an attribute. Therefore, this data needs to be preprocessed and export into a model where it can be clustered by different clustering algorithms. We decided following attributes of an event to be considered for clustering.

- Day of the week
- Minutes of the day
- Activity
- Class value

We have to normalize all the attributes so that it will not add any bias when calculating the cluster distance because of the attribute's scalar value. We select the range to be 0 to 1 (including 0 and 1) for every attribute.

The day of the week makes significant contribution when applying clustering [21]. Given an event, the day of the week attribute is assigned by mapping a value to each day such as Monday to 0, Tuesday to 1 etc. It is important to maintain the same order as the days of the week since it emphasizes the distance between each day. Once we map each day to a number, we normalize the value by the following expression so that the range is always [0-1]

$$day\_of\_week = \frac{mapped\_number\_of\_day}{6}$$

Minutes of the day is the amount of minutes passed from 12:00am of the day event occurred. Similar to other attributes we need to normalize the value to be range of [0-1] and it was done by following expression.

$$minutes\_of\_day = \frac{minutes\_passed\_from\_midnight}{1440}$$

Most recent activity is not associated with the event because there is no efficient way to identify what the user is doing at the time of the event. Therefore, what we did was, once we get notified that the user is changing the activity, we add an Activity type event and the timestamp. Then when we need to find the activity associated with the event, we looked back for the most recent Activity event and consider that Activity as the activity associated with the event. We configured the look back interval to be something like 10 minutes. If no Activity event is found within 10 minutes, we set the activity associated with the event as unknown. Similar to *day_of_week* we map each activity to a numerical value to normalize between [0-1]

Class value contains the type of the event and the primary data associated with the event such as contact name or opened application identifier. Class value is not used as a dimension/attribute for clustering but it need to identify which event is associated with that data instance.

3.2.2.2 Clustering and Predicting

Once the data is preprocessed and exported into the above model, it can be fed into different clustering algorithms. The output of the clustering operation is a data set of different clusters. For this research, we have selected K-means and EM clustering algorithms to be evaluated. Therefore these two algorithms are implemented and used for generating predictions.

In order to predict the user actions for a given time, what we need to do is creating an event similar to the events used in clusters. Then we can find the closest cluster for the newly created event and retrieve the dataset of the closest cluster. To find the closest cluster first we retrieve closest *n* number of events of each cluster comparing with the created event. Then we calculate the mean distance between the created event and the closest *n* number of events. The cluster which has the lowest mean value considers as the closest cluster.

By iterating through the closest cluster dataset, we can group the dataset entries by their class values. If we find the frequency of each class values in the closest cluster and sort the class values by their frequency descending order, we get a list of class values ordered by the most probable class value first. We can predict user action using these class values list and if we need *n* number of predictions, we can get the first *n* number of elements from this list.

3.2.2.3 Optimizations

An optional optimization can be done to improve prediction time if the cluster dataset entries can be sorted by a single numeric value and the numeric values of each dataset entry does not overlap between different clusters. In that case, we introduce a function like below to calculate numeric value for each dataset entry.

$$value = (100 \times activity) + (6 \times day\_of\_week) + (minutes\_of\_day)$$

Then find the minimum and maximum boundaries of each cluster by this value. If the boundaries of different clusters do not overlap, we can put the cluster into an ordered map and find the closest cluster by querying the floor value of the ordered map. The complexity of finding the value from a map is low and it will remove all the steps

which we have to do when finding the closest cluster. Additionally, we can get rid of cluster datasets and replace them with predicted user actions so that, any given time, we can directly retrieve the list of predictions. After analyzing clusters generated by two algorithms, we realized that clustered generated by K-means algorithm do not overlap and can be represented by a numeric value. Therefore, we have done this optimization only for K-means algorithm.

3.2.3 **Evaluation**

Since we selected K-means and EM clustering algorithms for this research, we mainly compared the accuracy between these algorithms. Apart from the accuracy, total clustering time, total prediction time, memory usage, was considered for evaluation. The total clustering time and the total prediction time are indications of CPU time. Therefore, by comparing accuracy, memory usage, total clustering time, total prediction time, we can find the best clustering algorithm which fits low end mobile devices with an acceptable accuracy.

Instead of using all the collected data for clustering, we separated the dataset into training and testing datasets. A randomly picked events can be used for testing data and they will not be included in the training dataset. By doing multiple tests with different testing datasets and calculating the average accuracy will remove the deviation of accuracy because of the noisy data. Since our goal is to come up with the best clustering algorithm to find predictions which are more frequent, we can add more frequent events to the test dataset while maintaining randomness.

We can measure accuracy by limiting first *n* predictions and test whether the test event is included with the *n* predictions or not. The accuracy percentage can be calculated as the following expression.

$$accuracy = \frac{number\ of\ correctly\ classified\ events}{number\ of\ events\ in\ test\ datase} \times 100\%$$

Memory usage can be captured using the IDE tools when running the tests. CPU time can be measured by capturing the total time it need to take for the clustering and total time it need to test all the events in the test dataset.

Under different scenarios the following metrics will be measured and considered for evaluation.

Table 3.1 Different test scenarios and the measured metrics

Scenario	Metrics		
	Accuracy	CPU Time	Memory
Run different clustering algorithms with different number of clusters	X	X	
Run different clustering algorithms considering temporal and activity as attributes	X	X	
Run different clustering algorithms considering only temporal attributes	X	X	
Apply possible optimizations and run different clustering algorithms	X	X	X

Since we have different set of datasets from different users, we ran the tests for each datasets and get an average value for accuracy. The results and the evaluation details are described in Chapter 5 – Results and Evaluation.

3.2.4 **Implementing the Framework**

The proposed framework should be self-contained and should have documented APIs to be consumed by developers. All the steps such as collecting data, preprocessing data, clustering and prediction must be contained within the framework.

The mobile application which used to collect data can be embed within the framework. The data collection background services can be reused directly. Since users actually need to access the contact numbers, the hashing method used in this application will not be suitable for the framework. Therefore, we have replaced the hashing with an encryption for contact numbers in the calls and SMS events. The encryption key was generated using a device unique identifier, so that it only be decrypted within the device. We selected AES encryption for this with a fixed

initiation vector. This makes encryption returns the output text for the same input text at any instance. This is important as it enabled developers to find the similarity of the events while not exposing the real contact number.

After evaluating and comparing the different results we can decide the best algorithm and appropriate parameters to be included within the framework. We configured to reset the clusters at the mid night of each day and form new clusters with new data. We implemented a method to get only the recent data if number of events exceeds a configured upper limit. This ensures that it does not exceed certain memory usage because of large data sets and ensures that only the recent data is used.

The API of the framework should hide the internal implementation as it is irreverent for the developer who uses this framework. The API should have methods to set which data need to be collected and how to retrieve the predictions. To pass data collection configurations, we created a user interface to be presented to the user and get the consent. This will ensure that 3rd party developers cannot enable data collection without the user consent.

When working with mobile programming it is necessary to have functions which do not block the main thread. The rendering of the user interface components are done by main thread and blocking the main thread will freeze the interface of the application which leads to a bad user experience. To overcome this issue, asynchronous functions are introduced for long running processes. In asynchronous functions, one function is provided to feed the data. This function will accepts the input data and does not return anything. It just sends the input data to a processor which will execute in a different thread other than the main thread. There will be a second function which works as a listener. Once the input data which feed to the first function is processed and ready, the second function will be invoked with the results. Therefore it is essential to expose asynchronous functions since predictions will take some time to process.

Apart from that if the developer needs to get the predictions from a background service, usual synchronous functions will be helpful. Synchronous functions will accepts the input data and do the processing in the same thread. It is important to

execute synchronous function only in a background thread so that it will not block the main thread.

For exposing these two types of functions, we have created a class which has following methods.

1. *processEvent()* & *onPredictions()* Listener – Asynchronous function *processEvent* which accepts a list of events and once the predictions are ready it will notify to the *onPredictions().* This is useful if the developer need to get the predictions in the main thread.
2. *processEventSynchronized()* – Synchronous function which accepts a list of events and returns the predictions. This is useful to get the predictions in a background thread.

CHAPTER 4

SYSTEM ARCHITECTURE AND IMPLEMENTATION

4. SYSTEM ARCHITECTURE AND IMPLEMENTATION

4.1 Overview

This chapter describes the implementation level decisions, approach of implementing data collecting mobile application and the proposed framework and their internal system architecture.

Before we start the implementation, we had to take some decisions which come across when developing a mobile application/framework.

- Selecting an operating system
- Selecting a runtime environment within the operating system

When selecting a mobile operating system to support our framework, the most important fact is that it should reach more user base. Currently, the most popular mobile operating systems are Android and iOS. Since Android is rapidly evolving and different mobile phone manufacturers embed Android as their operating system, Android has over 80% of the market share [19]. Additionally, Android exposes more low-level APIs to access user level data which is essential for the collection of user actions and events. Therefore, we decided to select Android as our target operating system.

Android has different runtime environments for mobile applications. The most common environment is Java runtime environment. This is also known as native runtime environment. There are some other runtime environments which build on top of Java runtime environment. Flutter, React native, IONIC are some of these environments and they even support cross platform (operating system) support. However, all these cross-platform supported runtime environments do not expose all the low-level APIs that native runtime supports. It is important to gather many data as possible for the success of this research and having an environment which exposes more low-level APIs would be ideal when gathering more data. Therefore, we decided to go with native runtime environment when developing this framework. Android has feature rich SDK to develop libraries, and mobile applications for native

runtime environment. Therefore, our framework can be distributed as an Android SDK library which the developers can include with their mobile applications.

From this point onwards, the proposed framework will be called as prediction framework. As described in the methodology chapter, apart from implementing prediction framework, we had to implement mobile application for collecting data as no publicly available dataset can be found for our requirement. This mobile application is designed so that the same components can be reused within prediction framework. Although there were some modifications needed to migrate the components, the architecture and the functionality remains the same.

4.2 Data Collecting Application

The data collecting application has following components shown in the Figure 4.1.

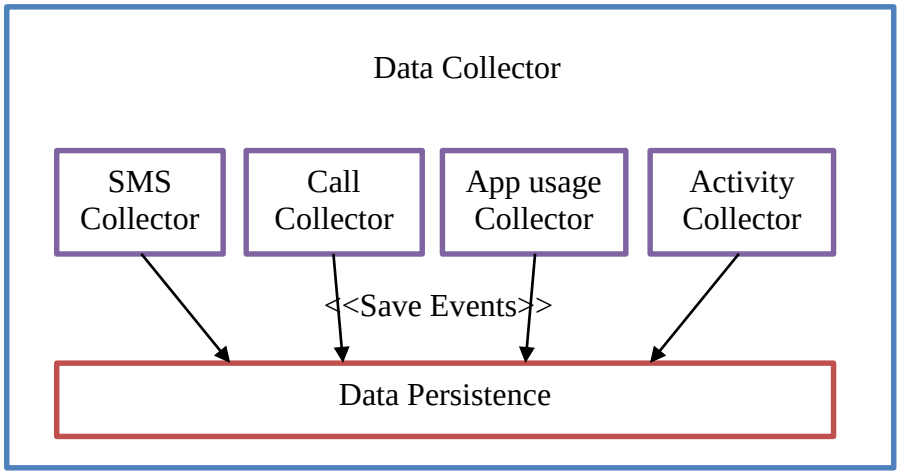


Figure 4.1 Data Collecting Application System Architecture

Data collector component is responsible for recording events from different sources. As mentioned in methodology chapter, we can use different types of sensors & software-based sources to record events in generalized manner. User activity, opened mobile applications, incoming & outgoing calls and received & send SMS messages were selected as the events for this research.

The collector components collect data from a single source and generate events. The generated events are passed to Data Persistence component to be persisted. Under this research scope we implemented 4 types of following collectors.

1. SMS Collector – Reads data periodically from SMS content provider given by the default message application. Then it generates events by adding received/sent timestamp and the contact phone number.
2. Call Collector – Reads data periodically from Call log content provider given by the default phone application. Then it generates events by adding time of the call and the contact phone number.
3. App Usage Collector – Reads App data periodically from operating system level usage stats manager APIs. Then it generates events by adding opening time of the app and the app package identifier.
4. Activity Collector – Reads the changes in the user activity using 3rd party library. Then generates events by adding the current time and the user activity type. User activity identification can be challenging and this research does not base on it. Therefore, an existing Google Awareness library is used [30]. This library notifies users activities like on foot, walking, running, on bicycle, and in vehicle.

Data Persistence component exposes *saveEvents()* function to persist the event. It is necessary to save events in a selected non-volatile storage. There are several data types and a SQL database would be the idea choice since individual events can be inserted and queried later. Android SDK inherently support SQLite databases and this implementation uses SQLite database to save passed events. The model of the event described in methodology chapter is recorded in a single table in the database with following columns.

- Action type – 1: Action cause by user, 2: Event caused automatically like incoming call, 3: Both action & event like a user activity.
- Event type – The type of the event. Values are ACT, APP, CALL, and SMS
- Timestamp –The date and the time of the event in timestamp format

- Class – The main action of the event. App package for APP type, phone number for CALL & SMS type, activity on ACT type
- Event data – Additional data.

For privacy reasons, phone numbers were hashed and encoded to base64 format before inserting into database. The Figure 4.2 shows sample data of different event types.

_id	action_type	event_type	event_time	data1	data2	data3	data4
1	2	SMS	1570023749584	87Msz7AmvOeaT+unN/LPWiX2G4bf1C4=	NULL	NULL	NULL
2	1	SMS	1568904289221	4XIh7o8TCiMsQWNLU64cPNdZhCbNbk3Dx...	NULL	NULL	NULL
349	1	CALL	1542639483110	MjqA9ZLRUwJhjHIXdtKXhKLOpg==	0	OUTGOING	NULL
350	2	CALL	1544274406688	TJ3k6Gj6Sg+lwjBU/1i2QZnlRLvxU1a47pa+	56	INCOMING	NULL
777	1	APP	1569422205861	com.viber.voip	NULL	NULL	NULL
782	1	APP	1569422214888	com.android.chrome	NULL	NULL	NULL
4552	3	ACT	1570370136713	WALKING	NULL	NULL	NULL
4553	3	ACT	1570370136728	ON_FOOT	NULL	NULL	NULL

Figure 4.2 Sample data of events database table

A background service was implemented to periodically upload the database file to a Firebase Storage so that we can download the files from different users. Different directories were programmatically created in the Firebase Storage for each device with a random identifier. The time of the upload is appended to the database file so that it will not any conflicts.

We have collected data from 20 mobile users throughout 1-month period. The selected users came from different occupations, countries so that it will add more variations to the data.

Figure 4.3 shows the main screen of the Data Collecting Application. The checkboxes give the users to select the data which they need to share. Start button runs the background service which triggers all the implemented collectors and the start the database file upload service. View button shows the raw data which was collected by the application.

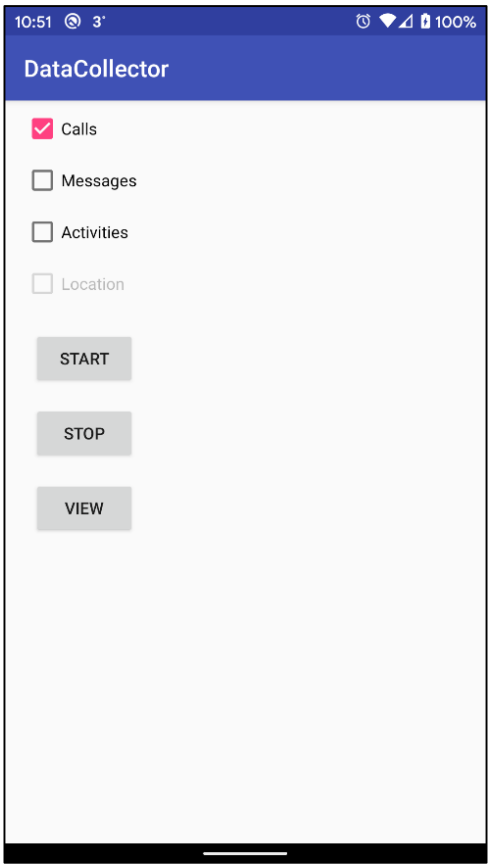


Figure 4.3 Data Collecting Application Main Screen

4.3 Prediction Framework

Figure 4.4 shows the high-level architecture of the Prediction framework. Arrows in the diagram shows the data flow direction between different components.

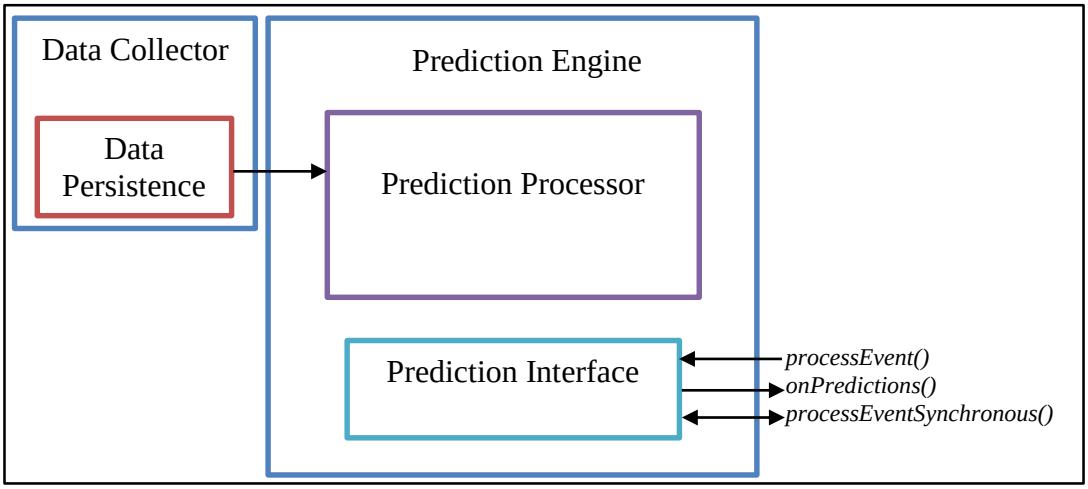


Figure 4.4 Prediction Framework System Architecture

4.3.1 **Prediction Engine**

Prediction Engine is responsible for retrieving data from data persistence component in the Data Collector component and send them to prediction processor component. Also, it exposes the framework APIs using Prediction Interface.

4.3.1.1 Prediction Processor

Prediction processor component preprocess the data first. We have created an interface named DataMapper which maps the raw events into clustering supported data model. There are two different implementations of DataMapper. They are important throughout the clustering process as they have the feature to convert event to clustering dataset instance and reverse that.

1. Time based data mapper – This creates the dataset instances with only temporal based attributes which are day of the week and minutes of the day.
   Parameter 1 – Day of week the event happened. Assigns a number from 0 to 6 and divide the value by 6 to get the range between 0 and 1
   Parameter 2 – Minute of the day the event happened. Value is divided by 1440 to get the range between 0 and 1
   Class – Class value of the event.
2. Time and Activity based data mapper – Apart from temporal based attributes, this mapper adds the user activity to the dataset instances. Given an event time, we fetched the most recent recorded activity as the user activity for the event. Since the user activity detection is not reliable, we limited fetching the

recent activity up to 10 minutes. This value kept as a parameter named lookbackMins in the data mapper.

Parameter 1 – Day of week the event happened. Assigns a number from 0 to 6 and divide the value by 6 to get the range between 0 and 1

Parameter 2 – Minute of the day the event happened. Value is divided by 1440 to get the range between 0 and 1

Parameter 3 – User activity when event happened. The mapped values are chosen so that similar activities have close values. Similar to other parameters the value are in the range of 0 to 1.

- ON_FOOT : 0.2
- WALKING : 0.3
- RUNNING : 0.6
- ON_BICYCLE : 0.8
- IN_VEHICLE : 1.0
- NO_ACTIVITY : 0.0

Class – Class value of the event.

```
public interface ClusteringDataMapper extends DataMapper {

    int getClassIndex();

    double generateKey(Dataset dataset);

    double generateKey(Event event);

    Event reverseKey(String entry);
}
```

Figure 4.5 Clustering Data Mapper Interface

Once the events are preprocessed using data mappers, it creates a temporary comma separated values (.csv) file. This file is directly feeded to the clustering algorithm. We have implemented two clustering algorithms K-means and EM which are configurable in Prediction Engine. Each algorithm has its prediction processor implementations. Having separate implementations for each algorithm gives the

freedom of adding optimizations and configurations based on the clustering algorithm.

Once the clustering is finished, prediction processor stores the clustered datasets. When it need to make predictions, it will find the closest cluster for the given input event. Finding closest cluster is described in 3.2.2.2 Clustering and Predictions section. After finding the closest cluster, the data mapper is used to reverse the conversion by transforming dataset instance into event. By collecting events and mapping class values of the events, we the predictions. The predictions are sorted by the frequency of the events.

4.3.1.2 Prediction Interface

Prediction interface is the public interface which exposes the functions to be called from integrated applications. When the functions are invoked, it calls the necessary functions in the prediction processor to return predictions. Figure 4.6 shows the interface of the Prediction Engine interface with Prediction Listener.

```
public interface PredictionEngine {

    List<Prediction> processEventSynchronous(Event event);

    void processEvent(Event event);

    interface PredictionListener {

        void onPredictions(List<Prediction> predictions);
    }
}
```

Figure 4.6 Prediction Engine Interface

4.4 Proof of Concept Application

A proof of concept mobile application named “Smart Suggestions” was developed by using the prediction framework library to demonstrate the usability of context based predictions. The basic settings of the application are to select which predictions the user expects through the application.

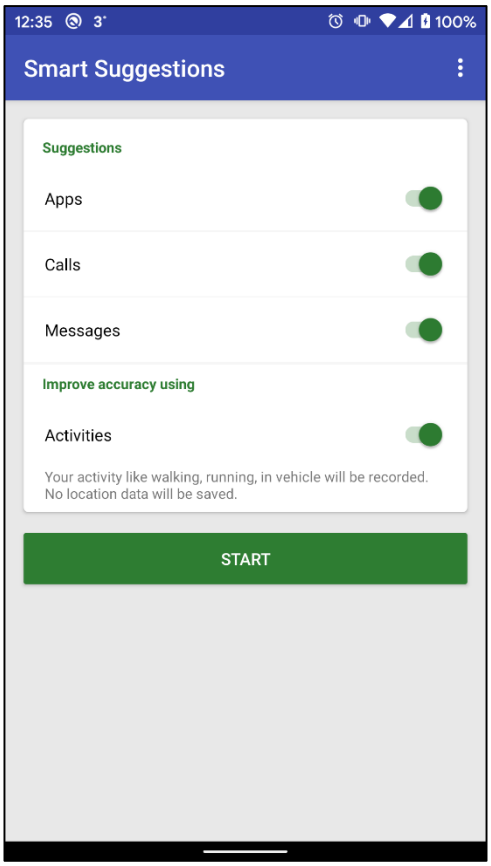


Figure 4.7 Proof of concept application settings

A background service was registered to detect the screen ON actions. Each time the user unlocks the screen, the application will get a call back from the background service. Then the application invokes the prediction interface method to get predictions by passing an event with the current time. The Prediction interface will return a list of predictions in order. If the predictions are available, the application shows a message bubble like icon to the user. If user taps that icon, the predictions will be shown. These predictions are categorized by their type (App usage, Call, SMS) and are presented in different tabs. The application limits the maximum number of predictions up to 5 per a category.

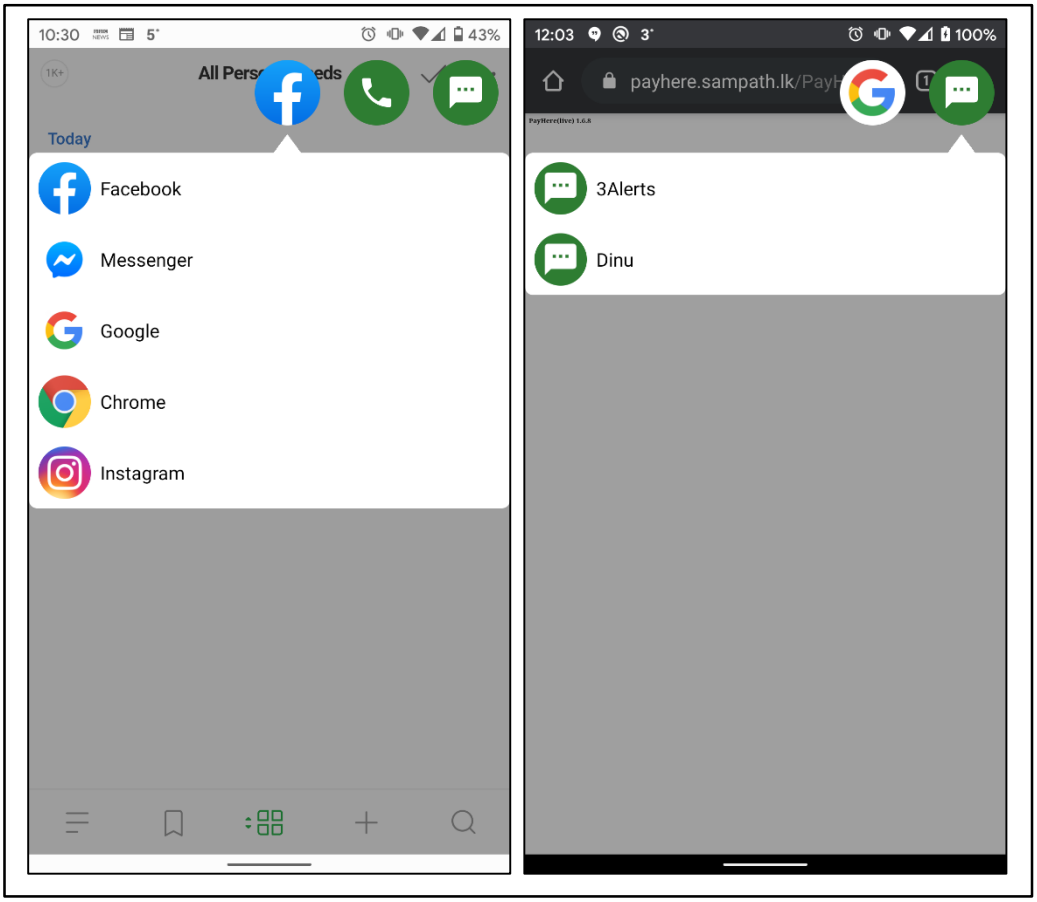


Figure 4.8 Application and SMS predictions screens

In Figure 4.8 left screenshot shows the application predictions. The right screenshot shows the SMS predictions. We have implemented showing following predictions.

1. Application predictions – Mobile applications name and icon will be presented in order of top item being the most frequent application for the given time and user context. Tapping a list item will launch the application
2. SMS predictions – A list of top contact numbers/names will be presented in similar order as application predictions. Tapping a list item will launch the default messaging application with conversation open for the selected contact.
3. Call predictions – A list of top contact numbers/names will be presented in similar order as others. Tapping a list item will launch the default call application with the phone number already filled to initiate the call.

Additionally, to demonstrate functionality and test the framework, Cluster Test screen and Cluster Accuracy screen was developed. In order to retrieve the data from the library framework, we exposed additional functions other than prediction interface functions. This was completely focused on debugging purposes and the production ready library will not include these additional functions.

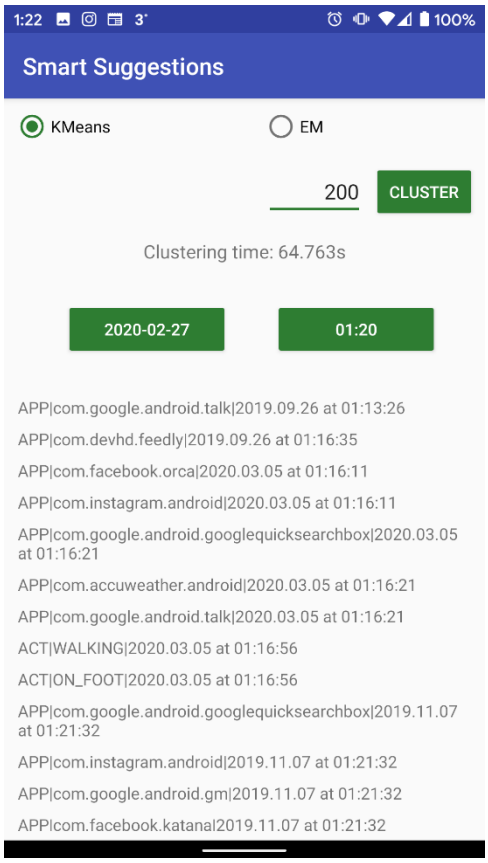


Figure 4.9 Cluster test screen

Figure 4.9 shows Cluster test screen. It has the option to select which algorithm (K-means or EM) to use and number of clusters needed. Once "Cluster" button is tapped, it will start the clustering process. Once done, clustering time is displayed. After the clustering the two buttons is there to select a date and time. Upon selecting date or time, the dataset of the closest cluster for the selection will be displayed.

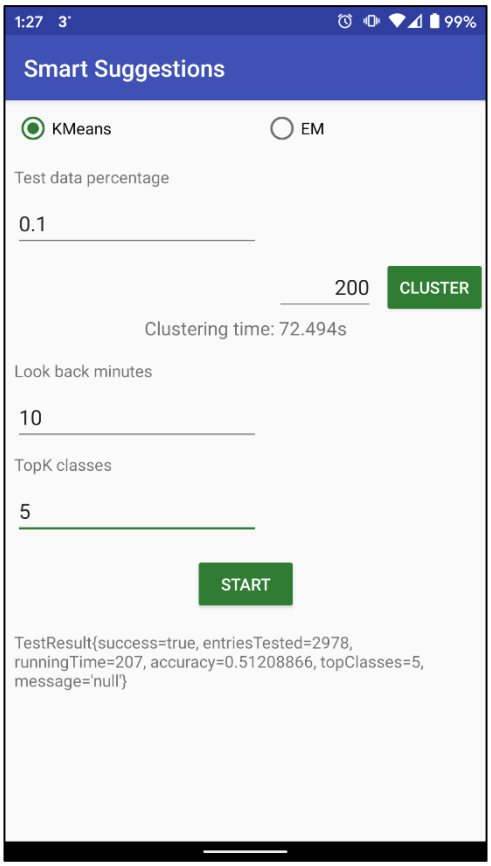


Figure 4.10 Cluster accuracy screen

Figure 4.10 shows the Cluster accuracy screen. This was used for evaluating different algorithms with different parameters. Similar to cluster test screen, this has the option to select a clustering algorithm. In accuracy testing we divide the dataset into two groups for training and testing. Training group was used for clustering and testing group was used for testing. The "Test data percentage" configures the proportion of the test dataset size out of all data. The "Look back minutes" configures the upper limit of finding the most recent user activity event to be consider as event's activity. The "TopK classes" configures the number of top predictions which need to be considers when testing. If the "TopK classes" value is 5, the topmost 5 predictions will be taken and check whether it matches with the test entity class.

CHAPTER 5

RESULTS AND EVALUATION

5. RESULTS AND EVALUATION

5.1 Overview

This chapter describes the nature of the tests, process of conducting them and the results. Later in the chapter we will talk about how the results are evaluated and how it contributed to the success of this research.

5.2 Testing Process

We have collected 20 datasets from different users and depending on their interactions with the phone number of data points varied from 5,000 to 40,000. The datasets which has lower than 10,000 data points were removed from testing as having small dataset does not reveal the patterns in the data. Out of 20 datasets 15 were acceptable and used to run different sets.

The results given in the rest of the chapter are average values unless we explicitly mentioned it. Averages are calculated by running the same test against selected datasets and divided by number of datasets.

For all the tests except parameter tuning tests, we used a Google Pixel 2 phone which has octa-core CPU (4x2.35 GHz Kryo & 4x1.9 GHz Kryo) and 4GB RAM. For parameter tuning tests we used a Dell Inspiron 13-7386 laptop having octa-core Intel® Core™ i7-8565U CPU (8x1.80 GHz) and 16 GB RAM.

5.2.1 **Parameter Tuning**

Both K-means and EM clustering algorithms need the number of clusters which need to break when applying the algorithm. There is no universal way of deciding the value for this application. Therefore, we tested both algorithms with different data mappers (time only mapper and time with activity mapper) with variations of cluster count.

The Table 5.1shows the accuracy percentages of different scenarios against changes in the input cluster count. Note that when cluster count is 200, the maximum

accuracy achieved in every test. Therefore we decided set 200 as cluster count for other tests.

Table 5.1 Accuracy percentages for different cluster counts

Cluster Count	K-means		EM	
	Time only	Time+Activity	Time only	Time+Activity
100	58.18	62.43	66.06	71.40
150	65.66	70.32	73.58	76.58
200	69.36	73.09	75.15	81.01
250	67.28	70.19	74.46	77.54
300	63.24	68.19	69.22	73.49
350	59.92	63.19	64.37	68.14
400	55.96	60.70	60.86	65.54
600	50.40	52.48	57.85	61.83

Table 5.2 contains the clustering time in seconds on different test scenarios. These tests were done using a personal computer and the results may differ in a mobile phone processor.

Table 5.2 Total clustering time in seconds for different cluster counts

Cluster Count	K-means		EM	
	Time only	Time+Activity	Time only	Time+Activity
100	9.29	8.09	58.86	46.27
150	11.61	18.05	78.55	63.57
200	18.77	30.37	94.32	70.55
250	19.51	34.27	110.58	78.42
300	33.25	53.03	126.10	87.73
350	43.74	70.85	133.94	106.53
400	53.65	116.27	139.29	125.68
600	92.39	228.45	163.53	149.37

5.2.2 **Optimization Testing**

The optimization discussed in section 3.2.2.3, is applicable only to K-means algorithm. To compare the effect of this optimization, we have measured the average clustering time, average testing time, memory consumption and the accuracy between regular EM clustering, regular K-means and the optimized K-means implementations.

Table 5.3 Results comparison between different clustering algorithms

Metric	K-means	K-means optimized	EM
Average clustering time (s)	139.38	150.35	774.45
Average testing time per entry (ms)	19.24	0.08	21.34
Accuracy (%)	74.95	72.38	79.31
Memory consumption (Mb)	17	4	16

Average testing time per entry is calculated using total time it took to evaluate the test dataset and the size of the test dataset. The memory consumption was measured using the difference between before and after running the clustering. At the each run, forced garbage collection was triggered when capturing the memory usage.

Figure 5.1 shows the memory usage of when clustering and testing a small dataset when applying the regular K-mean clustering. The red dots on top marks the starting of the clustering and the testing respectively. Figure 5.2 shows the same when applying optimized K-means clustering. The end of each figure marks the end of the testing.

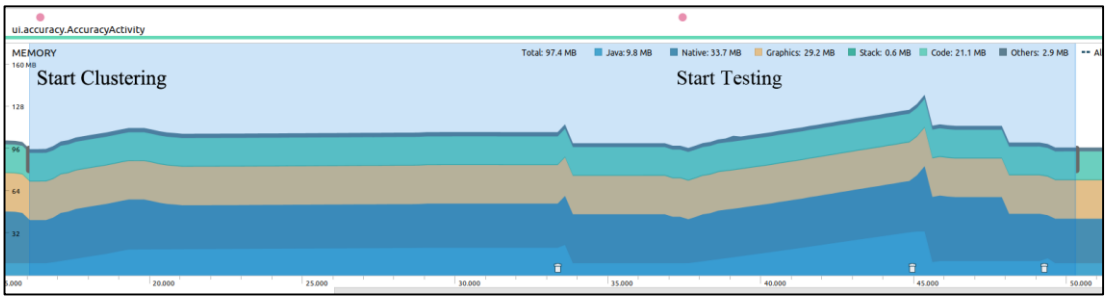


Figure 5.1 Memory usage when regular K-means clustering and testing

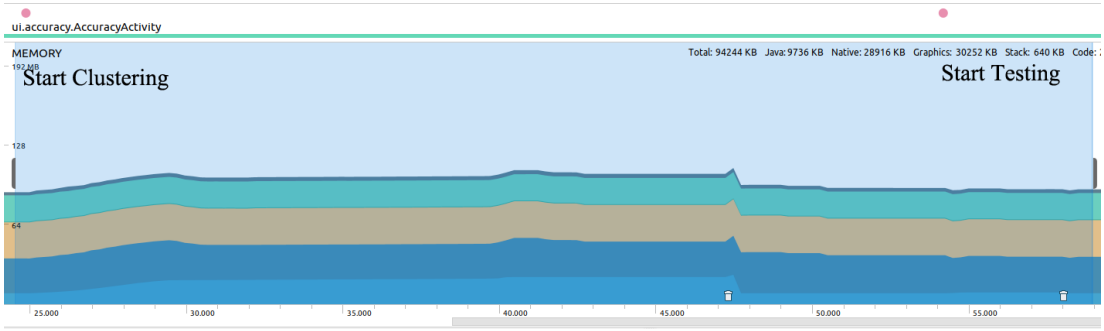


Figure 5.2 Memory usage when K-means optimized clustering and testing

5.3 Evaluation

Figure 5.3 is a graph plotted using the data in accuracy percentage Table 5.1. We can clearly identify that the highest accuracy reached when clustering count is 200. Having too many clusters makes over-fitting the data. On the other hand, having few clusters makes under-fitting. It is essential to find an approximate optimal cluster count for achieving high accuracy.

Also, we can identify that EM clustering is more accurate than K-means in most of the scenarios. This is because K-means clusters tend to create approximately equal time ranged clusters while EM clustering tend to find more patterns.

The other finding is that by adding user activity improves the overall accuracy on both algorithms. This depicts that the more we use context data, accuracy will increase. This behavior is expected because having more dimensions in data points, will create more fine-grained clusters which results identifying better patterns.

Under this model, we can go up to ~80% accuracy by using EM clustering with temporal and user activity data. Using K-means clustering will give an accuracy up to ~74%. In this case, EM clustering is the best candidate when it comes to achieving good accuracy.

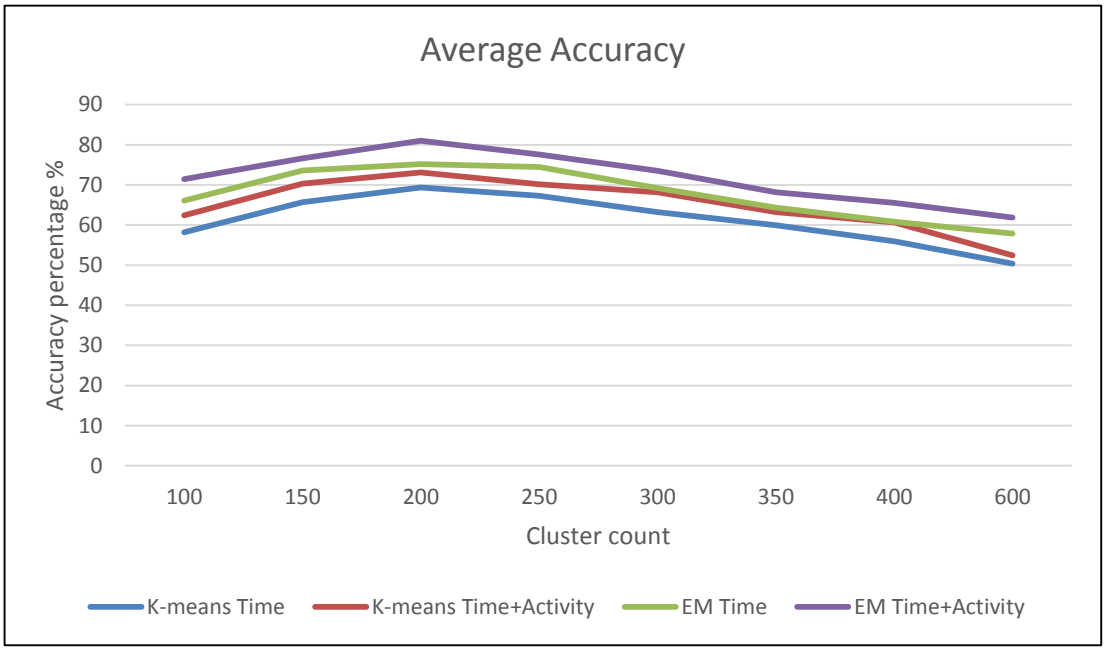


Figure 5.3 Different average accuracy percentages

Table 5.2 indicates that EM clustering takes considerably more CPU time when the cluster count is low. However, it performs when the cluster count is getting higher. But our approximate optimal cluster count is not higher enough for EM clustering to catch the CPU time of K-means. In our case, K-means is faster, and is more suitable for mobile devices as mobile devices do not have high end CPUs. The framework is not designed to perform clustering frequently. The framework was designed to cluster at the very beginning or at the midnight of each day. Therefore, having a high CPU time for clustering is acceptable within the scope of the framework. The only drawback in high CPU time is, it may not suit low-end devices.

By analyzing the data in Table 5.3, we can clearly identify the significant improvement in testing time in the optimized K-means clustering. This is important as time taken for a prediction is highly improved. However, we can see that there is a

nearly 2% reduction in the accuracy of the optimized implementation. Unlike finding the closest cluster by using the real dataset points, optimized implementation checks only the boundaries of the clusters. Therefore, there is a chance of not picking the optimal closest cluster in the optimization. The 2% accuracy reduction is acceptable for an application where the framework needs to make higher amount of predictions. There is an increase in CPU time, but it is not significant. Although the clustering time is equal, the initiation of predictions from all clusters adds extra time.

Comparing Figure 5.1 and Figure 5.2 reveals that testing on regular K-means implementation consumes more memory when testing. It is because of finding closest cluster needs creating intermediate objects. Apart from that comparing memory usage in Table 5.3 we can see that optimized implementation need less memory to keep the results. Because the regular implementation needs to keep all clusters for identifying the closest cluster, it needs more memory. But optimized implementation does not need all the clusters because it only stores the predictions which is aggregated data from all the clusters. By considering above factors, optimized implementation is suitable for low-end devices because of low memory usage and acceptable accuracy.

CHAPTER 6

CONCLUSION

6. CONCLUSION

With the evaluation of mobile phones, some people tend to use their mobile phone even more than an hour a day. Throughout the day, people interact with the smartphones continuously, but their software is not that smart to identify what the user's intentions are. For an example if the user is driving, there is a high chance of opening a navigation/maps app. But the current software does not try to make that easy for the user. The problem is that there is no underlying method for the mobile application developers to identify the user context and get user's intentions so that it will make user experience better.

6.1 Research Contribution

This research answers the above problem by implementing a framework so that mobile application developers can embed and get predictions on what user's next action is.

In literature review we have found that there are several methods to use sensor data to identify user activity and there are production ready libraries already available to use. We also analyzed that software-based data sources like SMS messages and call logs have a good potential to help identifying user's context. Considering these factors, we have designed a generalized model to collect and persist use activities, SMS and call logs.

Comparing properties of different pattern recognition methods, we decided that clustering is a good fit for running within mobile devices as it does not need high computation power which is a restriction in mobile devices. Also, it resolves the requirement of making this work offline.

By using collected data from different users, we have evaluated K-means and EM clustering algorithms and even we have found an efficient way to make prediction if they come from K-means algorithm based clusters. By analyzing the research results, we conclude that EM clustering gives high accuracy and it is suitable for high-end devices while optimized K-means clustering is suitable for low-end devices with less accuracy but more compatibility.

The framework was developed as an Android library so that any Android mobile application developer can import and use the library functions easily. Further, we developed a proof of concept application which uses the above library and predict user's mobile application openings, calls and SMS message contacts so that user do not have to go through the application launchers to select required application.

6.2 Limitations

6.2.1 Scalability

The accuracy increases as the collected data increases, but it will make the clustering slow and resource intensive. As a temporary solution we have limited number of events when clustering, but we need a scalable solution where we can include all the collected data to improve the accuracy.

6.2.2 Accuracy

We have achieved 80% accuracy level by using EM clustering. But it always has the chance to increase the accuracy more. Although our research approach covers temporal patterns, it does not vary predictions based on the chain of actions. For an example if the user tends to open app 'A' followed by app 'B', this research solution does not predict opening app 'B' unless there is a past event at the same time.

6.2.3 Privacy

Collecting user related data is always connected with privacy concerns. Even though the framework was designed not to upload or expose data, there is a chance of revealing user information even from the encrypted data.

6.3 Future Research Directions

From the very beginning of this research, one requirement is to make this work irrespective of a network connection. This restricted us from using complex or high computational approaches like neural networks which may give better accuracy levels. As an example, having long short-term memory (LSTM) based neural network will give better accuracy since it can maintain user context as well as previous actions when making predictions. As the neural network frameworks evolve, some of them support running lightweight version of neural network models

within the mobile device. This may open new doors to extend this framework by adding new neural network based prediction processors.

Since mobile device capabilities are constrained by low CPU and memory capacities, it can use computation offloading techniques to run complex prediction models. There are many researches done related to mobile cloud computing [31] which can be beneficial for improving accuracy and response time of the framework.

7. REFERENCES

[1] M. Lee, C. Bak, and J. W. Lee, “A prediction and auto-execution system of smartphone application services based on user context-awareness,” J. Syst. Archit., vol. 60, no. 8, pp. 702–710, 2014.

[2] W. P. Lee and K. H. Lee, “Making smartphone service recommendations by predicting users’ intentions: A context-aware approach,” Inf. Sci. (Ny)., vol. 277, pp. 21–35, 2014.

[3] S. A. Hoseini-Tabatabaei, A. Gluhak, and R. Tafazolli, “A Survey on Smartphone-Based Systems for Opportunistic User Context Recognition,” ACM Comput. Surv., vol. 45, no. 3, p. 27:1–27:51, 2013.

[4] A. Misra and L. Lim, “Optimizing sensor data acquisition for energy-efficient smartphone-based continuous event processing,” Proc. - IEEE Int. Conf. Mob. Data Manag., vol. 1, pp. 88–97, 2011.

[5] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, “Activity Recognition Using Cell Phone Accelerometers,” SIGKDD Explor. Newsl., vol. 12, no. 2, pp. 74–82, 2011.

[6] G. Maggiore, C. Santos, and A. Plaat, “Smarter smartphones: Understanding and predicting user habits from gps sensor data,” Procedia Comput. Sci., vol. 34, pp. 297–304, 2014.

[7] F. Liu, D. Janssens, J. Cui, G. Wets, and M. Cools, “Characterizing activity sequences using profile Hidden Markov Models,” Expert Syst. Appl., vol. 42, no. 13, pp. 5705–5722, 2015.

[8] T. M. T. Do and D. Gatica-Perez, “Contextual conditional models for smartphone-based human mobility prediction,” Proc. 2012 ACM Conf. Ubiquitous Comput., p. 163, 2012.

[9] J. Ye, S. Dobson, and S. McKeever, “Situation identification techniques in pervasive computing: A review,” Pervasive Mob. Comput., vol. 8, no. 1, pp. 36–66, 2012.

[10] P. Dai, S. S. Ho, and F. Rudzicz, “Sequential behavior prediction based on hybrid similarity and cross-user activity transfer,” Knowledge-Based Syst., vol. 77, pp. 29–39, 2015.

[11] Y. Kawahara, H. Kurasawa, and H. Morikawa, “Recognizing user context using mobile handsets with acceleration sensors,” In Proceedings of the IEEE International Conference on Portable Information Devices (PORTABLE’07). pp. 1–5, 2007

[12] D. Siewiorek, A. Smailagic, J. Furukawa, A. Krause, N. Moraveji, K. Reiger, J. Shaffer, and F. L. Wong, “SenSay: A context-aware mobile phone,” IEEE International Symposium on Wearable Computers. pp. 248–249, 2003

[13] J. Yang, “Toward physical activity diary: Motion recognition using simple acceleration features with mobile phones,” International Workshop on Interactive Multimedia for Consumer Electronics. pp. 1–10, 2009

[14] T. Brezmes, J. Gorricho, and J. Cotrina, “Activity recognition from accelerometer data on a mobile phone,” International Work-Conference on Artificial Neural Networks: Part II: Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living. pp. 796–799, 2009

[15] I. König, B. N. Klein, and K. David, “On the stability of context prediction,” Proc. 2013 ACM Conf. Pervasive ubiquitous Comput. Adjun. Publ. - UbiComp ’13 Adjun., pp. 471–480, 2013.

[16] A. Bastawrous, “mHealth Possibilities in a Changing World. Distribution of Global Cell Phone Subscriptions,” J. Mob. Technol. Med., vol. 2, no. 1, pp. 22–25, 2013, doi: 10.7309/jmtm.78.

[17] “Internet access – households and individuals, Great Britain: 2018”. [Online]. Available: https://www.ons.gov.uk/peoplepopulationandcommunity/householdcharacteristics/homeinternetandsocialmediausage/bulletins/internetaccesshouseholdsandindividuals/2018 [Accessed 20-Feb-2020].

[18] P. Pelegris, K. Banitsas, T. Orbach, and K. Marias, “A novel method to detect heart beat rate using a mobile phone,” 2010 Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBC’10, no. February 2014, pp. 5488–5491, 2010.

[19] "Smartphone Market Share | OS Data Overview ". [Online]. Available: https://www.idc.com/promo/smartphone-market-share/os [Accessed 24-Feb-2020].

[20] N. Singh and D. Singh, “Performance Evaluation of K-Means and Heirarichal Clustering in Terms of Accuracy and Running Time,” Int. J. Comput. Sci. Inf. Technol., vol. 3, no. 3, pp. 4119–4121, 2012.

[21] D. Kondor, S. Grauwin, Z. Kallus, I. Gódor, S. Sobolevsky, and C. Ratti, “Prediction limits of mobile phone activity modelling,” R. Soc. Open Sci., 2017.

[22] "Android Sensor Types". [Online]. Available: https://source.android.com/devices/sensors/sensor-types [Accessed 26-Feb-2020]

[23] V. K. Singh, L. Freeman, B. Lepri, and A. Pentland, “Predicting spending behavior using socio-mobile features,” Proc. - Soc. 2013, no. September, pp. 174–179, 2013.

[24] R. Sathya and A. Abraham, “THE SCIENCE AND INFORMATION ORGANIZATION Editorial Preface,” Int. J. Adv. Res. Artif. Intell. Int. J. Adv. Res. Artif. Intell., vol. 2, no. 2, pp. 34–38, 2013.

[25] U. Christoph, K. H. Krempels, J. Von Stülpnagel, and C. Terwelp, “Automatic context detection of a mobile user,” WINSYS 2010 - Proc. Int. Conf. Wirel. Inf. Networks Syst., no. August, pp. 189–194, 2010.

[26] D. Peebles, H. Lu, N. D. Lane, T. Choudhury, and A. T. Campbell, “Community-guided learning: Exploiting mobile sensor users to model human behavior,” in Proceedings of the National Conference on Artificial Intelligence, 2010.

[27] M. Berchtold, M. Budde, D. Gordon, H. R. Schmidtke, and M. Beigl, “ActiServ: Activity recognition service for mobile phones,” in Proceedings - International Symposium on Wearable Computers, ISWC, 2010.

[28] A. Y. Ng and M. I. Jordan, “On discriminative vs. Generative classifiers: A comparison of logistic regression and naive bayes,” in Advances in Neural Information Processing Systems, 2002.

[29] “A Gentle Introduction to Expectation-Maximization (EM Algorithm)”. [Online]. Available: https://machinelearningmastery.com/expectation-maximization-em-algorithm/ [Accessed 01-Jan-2020]

[30] “Google Awareness API | A unified sensing platform enabling applications to be aware of multiple aspects of a user’s context, while managing battery and memory health”. [Online]. Available: https://developers.google.com/awareness [Accessed 01-Mar-2020]

[31] K. Akherfi, M. Gerndt, and H. Harroud, “Mobile cloud computing for computation offloading: Issues and challenges,” Appl. Comput. Informatics, vol. 14, no. 1, pp. 1–16, 2018, doi: 10.1016/j.aci.2016.11.002.