EXPLORING A MACHINE LEARNING BASED APPROACH FOR ANALYZING ANONYMIZED DATA

by

Derek Nheiley

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
April 2017

This thesis is dedicated to my loving wife and parents who supported my pursuit of higher education.

Table of Contents

List of Tables

List of Figures

Abstract

Information found in log files is often stored in human readable plain text. Research study participants do not want sensitive information recorded, and when data is made publicly available, participants ask that they cannot be individually linked to their data [22][10]. Researchers anonymize publicly available datasets using randomly assigned encodings for users, and one-way hashing functions for encrypting other human readable plain text.

This research examines the effects of concatenating and hashing a list of nominal values to represent a single dataset feature, using password encryption as an example. Using decision trees and classification accuracy as a measure of information leakage, I evaluate the performance on several publicly available mobile datasets.

One of the contributions in this research identifies fine grain application usage details as a suitable candidate for device fingerprinting, which maintains user classification accuracy even when obscuring the name and number of applications.

Glossary

anonymization	Type of information sanitization whose intent is privacy protection. 1, 3–5, 24–27, 44–46
digest	The output from a hash function, also known as a hash. 7, 8
FHE	Fully Homomorphic Encryption. 3
hash	The output from a hash function, also known as a digest. 7, 8, 26, 27, 39, 44–46
hash function	A one way function which maps an input value to an output value based on a given hashing algorithm. 1, 2, 7, 8, 26, 45
MAC [address]	Media access control address of a computer is a unique identifier assigned to a network interface. 13, 23
SQL	Structured Query Language. 17
SSID	Informal (human) name of the basic service set in a communication network. 13, 18, 19, 23, 26
TSV	Tab separated value. 13, 32

Acknowledgements

Thanks to all my academic examples such as my supervisor, professors, instructors, teaching assistants, and classmates for providing a stimulating learning environment and a challenging but rewarding path to follow towards all my future problem solving endeavours.

I would also like to thank 2Keys Corporation for the opportunities created by the NSERC program.

Chapter 1

Introduction

Often when log files are publicly available, information that is human readable plain text, such as usernames, application names and wifi access point names are anonymized. The process of anonymization can be categorized in two main categories for this research, encoding and encrypting.

Encoding (also known as renaming, or indexing) is the process of transforming values using some type of legend, or key. Values in a dataset feature such as the username “Alan Turing” could be encoded to “user4”, or more simply the number ‘4’. Here the output value can be randomly generated, as long as it is not already used by some other value, but otherwise is not affected by the actual input value. In order to add more data after encoding a dataset feature is complete, a copy of the legend or key must be re-used, it lists all the input values, and the randomly generated output value they should be replaced with. New values are then encoded using what ever mapping is listed so that the same original data corresponds to the same encoded data. However, the legend, or key is rarely provided when public data has been encoded, because that would eliminate any gains in privacy achieved by encoding the dataset. Therefore, adding new data, or combining multiple datasets that where each individually encoded is not useful.

Similar to how passwords can be stored, plain text can be encrypted (among other ways[1]) using a one way function which transforms the text into scrambled looking output which cannot easily be transformed back into the original text (discussed further in chapter 3: hash functions). The advantage of encrypting data using a one way function is that the same input will always result in the same output, so more data can be added later using the same process because the encryption method is often publicly stated. When performing analysis on the data, encoding (usually in the form of indexing with a new number for each unique value seen) is then applied

[1]Symmetric and public key cryptography are also popular methods for encrypting data.

to the scrambled outputs immediately before the dataset is processed.

The ability to add new data when datasets are encrypted using one way functions is also its primary weakness. For example, if the plain text “password” was encrypted using a one way function, and a dataset instead stored “a1b2c3d4”, I could try and guess what plain text was used to generate “a1b2c3d4” by inputing my guess to the same one way function and comparing the output values[2]. For features such as “Application Name”, lists of popular applications can easily be collected from online stores and used to guess the encrypted values in publicly available datasets.

Continuing previous research on mobile log files where users could be identified by seemingly benign device log files [19] this research investigates how users can be identified even when hashing is used to anonymize and encrypt log files in different ways. Specifically, I propose a method where variable length nominal values from a dataset feature are first concatenated to a single value, and then encrypted as a single value to generate an encrypted output which obscures both the length, and plaintext values and the nominal values for an individual dataset feature. I then evaluate how information leakage is affected compared to a baseline experiment.

The remainder of this thesis is organized as follows. Chapter 2 discusses the background and subsequent motivations for this research with comparison to related works. Chapter 3 provides a technical background on existing methods such as hash functions and decision trees. Dataset descriptions are provided along with some detailed analysis about differentiating factors. After outlining a baseline method, the proposed method is shown with examples. I have presented the challenges experienced during this research in chapter 4. First discussing the effects of the different datasets, and how overfitting affects this research. Chapter 5 presents and discusses the results including how the datasets are compared with different numbers of users. Finally chapter 6 draws conclusions and outlines options for future work.

[2]The process of using a list of guess inputs to generate outputs from a one way function and comparing the results to the output that is guessed is called a dictionary attack.

Chapter 2

Background

Machine learning on encrypted data often falls into two broad categories, privacy-preserving training, and privacy-preserving classification [6]. Under privacy preserving training, only the server should know the training data, and construction of the predictive model, where as, privacy-preserving classification, the inputs and prediction are known only to the client. In both approaches, the inputs into the model can be encrypted using fully homomorphic encryption [6].

Until 2009 [11], homomorphic encryption had several limitations such as the number of operations, or runtime performance. Fully Homomorphic Encryption (FHE) works with an arbitrary number of operations, and in some implementation, mitigates the performance penalties [12].

In contrast to recent FHE advances, data mining has been exploring methods for privacy preserving classification for ten or more years before FHE. Concepts such as k-anonymity are still aiming to strike a balance between anonymization and data mining utility [19] [23].

Mining boolean data values under limited interaction model of privacy-preserving classification was presented by Yang *et al.* [27], where they defined privacy in a semi-honest model stating: "*no extra information about honest customers values be leaked even if the miner receives help from corrupted customers.*" [27] Yang also discusses the randomization technique [27] further detailed by Du *et al.* [7], where Du explores storing $X_i + r$ in a database (where X_i is a private data value, and r is a random number from a known distribution).

2.1 Motivations

Anonymized log files are offered on many devices as a method of anonymously sending usage information to application developers. Many operating systems and applications offer options to send log files when recovering from errors in addition to sending regular usage information such as system load and memory state.

Just as encrypted network traffic can still be differentiated using flow analysis [13], this research investigates how compressing and encrypting log files on mobile devices can affect the potential for being identified as an individual or user type.

Further to the discussion of restrictive access of third party applications to system data by Shepard *et al.* [22], and the use of hashing for preserving privacy in logs files, the method proposed in this research may viewed as a potential basis for an anonymized log file API where the hardware or operating system provides anonymized information to third party applications on security conscious devices [22].

2.2 Related Work

Previous research has examined information leakage and device fingerprinting, and have demonstrated the possibility of information leakage, but they focused either on mobile specific setting or attacks on encryption, and not both.

2.2.1 Is This You? Identifying a Mobile User Using Only Diagnostic Features

Quattrone *et al.* [19] detail their success achieving %94 accuracy in identifying users by way of a Naive Bayes classifier on hardware statistics and system settings from mobile device logs. While Quattrone *et al.* summarize daily information as part of their pre-processing, they do not directly attempt to use any type of anonymization, encryption, or global recoding [26] to investigate the effects of intentionally obscuring original data values.

2.2.2 Data Driven Authentication: On the Effectiveness of User Behaviour Modelling with Mobile Device Sensors

The GCU datasets used in this research (see section 3.2.1) originated from Glasgow Caledonian University in relation to the work carried out by Kayacik *et al.* in 2014 [10]. In their above named publication, Kayacik *et al.* propose "*a lightweight, and temporally and spatially aware user behaviour modelling technique for sensor* *based authentication*" with the goal of investigating "*practical aspects*" [10] of incremental training duration, automatic deployment on a per user basis, which allows for variations in performance between users and addresses the impacts of behaviour drift over time.

Using several statistical models based either on temporal or spatial data allowed Kayacik *et al.* to "*build comfort levels based on multiple indicators*", build the model incrementally, and on the device because of their low computational and storage costs.

Discussing the background and motivations for implicit authentication, Kayacik *et al.* note that as of 2013, "*64% of users do not use authentication on their phones*" [10][5]. The topic of security and privacy is further implied when they comment to the benefit of a lightweight user modelling approach, "*some users may prefer an on-device modelling technique that allows them to build and deploy models without their data ever leaving the device*" [10].

Focusing on the information security aspect of the previous statement about data leaving a mobile device, this research will aim to provide an option for anonymizing data, while still preserving the entropy for user classification. Kayacik *et al.* also note the importance of spatial information [10] to user behaviour and its contribution to signalling deviations from baseline user behaviour on mobile devices. I will continue to employ spatial information throughout my experiments comparing the results before and after my proposed anonymization method.

2.2.3 LiveLab: measuring wireless networks and smartphone users in the field

The RICE dataset also used in this research (see section 3.2.3) was made publicly available by the LiveLab Project out of Rice University in 2010 [2]. Shepard *et al.* state that one of their primary concerns was privacy, and based on pre-observation interviews, participants stated that their "*biggest concern*" was they did "*not want researchers to be able to associate their identities with their data*" [22].

Although participants stated they were "*not concerned about some potentially sensitive data being collected as long as the data is not directly linked to their identity*", the researchers chose to use "*one-way hashing to preserve the uniqueness of the data* *entry without revealing its content*" [22]. In certain implementations, the use of hashing can leave data vulnerable to dictionary attacks (discussed later in section 3.1.1).

The LiveLab methodology for privacy protection using hashing was one of the motivations for this research. Shepard *et al.* also demonstrate the importance of "*temporal dynamics and trends of application usage*" [22] which I attempt to support in my proposed method of anonymizing categorical data values.

2.2.4 Device Fingerprinting

"*Device fingerprinting is the process by which a device or the software it is running is identified by its externally observable characteristics*" [8]. Franklin *et al.* discuss wireless device driver fingerprinting, and although the concept of device fingerprinting is not new, they discuss the details of their "*timing based approach to when observing implementation dependent differences ... [in]... inter-frame timing of transmitted probe requests*" [8].

Similar to log files, Franklin *et al.* note that "*coarse grained timing information is preserved despite the encryption of frame content...*" [8]. A common approach to anonymizing log files often ignores timestamps, leaving them in original form. Unfortunately as discussed in section 3.2.4, the datasets used in this research are collected primarily on set intervals. I will come back to the concept of device fingerprinting when I identify other potential sources in section 3.2.2 that are also preserved even when encrypted.

Chapter 3

Methodology

Existing methods such as hash functions and decision trees are reviewed in this chapter before going into a detailed description of the datasets used in this research. After outlining a baseline method to compare results, the proposed method is described, including examples of its usage on the datasets used in this research.

3.1 Algorithms

Part of the investigation undertaken in this research involves several types of algorithms which will now be reviewed.

3.1.1 Hashing Functions

Hashing is the one way process of mapping a variable length input to a fixed length output (known as the digest, or hash) using a reproducible function (known as a hash function).

$$X_{digest} = hashFunction(X) \tag{3.1}$$

Hashing first appeared in the early 1950's, as detailed by Alan G. Konheim [16], although at the time hashing was known as "scatter storage". The term, "hashing" appeared in relation to hash functions until several articles in 1968 [17] [18] [16].

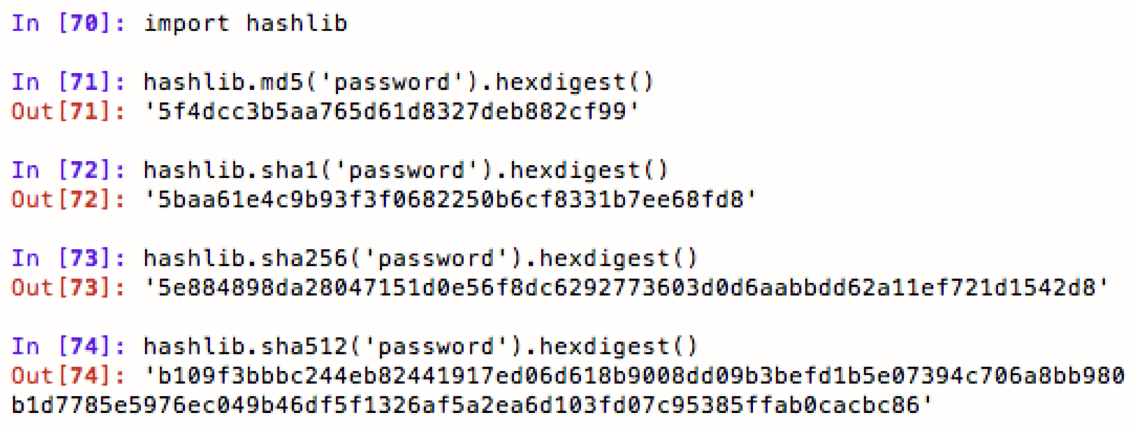
Often hashing involves mapping a block of data from a large input space to a specific value in the smaller output space. Because the input space is often larger than the output space, two input values may map to the same output value, this is known as a hashing collision. Depending on the particular implementation of the hashing algorithm used, collisions can present a security concern.

For example, authentication systems often store the hash of a password instead of the plaintext version. If “password” maps to the hash “a1b2c3d4” (for some pre-defined hash function), then the authentication system stores “a1b2c3d4” and applies the same pre-determined hash function to any input and checks to see if it matches the hash stored. The problem arises when some input such as “aoiwenelcl” also maps to the same hash “a1b2c3d4”. In this case, the authentication system will also accept “aoiwenelcl” as a valid password.

As hashing algorithms have progressed over the years, various measures have been implemented to prevent collisions. In programming languages that implement hash tables as key→value data structures, the original input key can be stored along with the hash of the input key in order to differentiate collisions and return the correct value for a given input key when retrieving a value from the data structure.

In security settings, transmitting and storing the plaintext version of the input to a hash function has various security concerns such as man-in-the-middle attacks and data breaches just to name a few. Instead hashing algorithms often increase the size of the output space in an attempt to reduce the chance of collisions, along with other complexities internal to their algorithm.

An example of using a hashing library in python is show in figure 3.1, note the length of the digest (shown in hexidecimal format) increases for newer hash algorithms such as SHA256 and SHA512.



```
In [70]: import hashlib

In [71]: hashlib.md5('password').hexdigest()
Out[71]: '5f4dcc3b5aa765d61d8327deb882cf99'

In [72]: hashlib.sha1('password').hexdigest()
Out[72]: '5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8'

In [73]: hashlib.sha256('password').hexdigest()
Out[73]: '5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8'

In [74]: hashlib.sha512('password').hexdigest()
Out[74]: 'b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980
b1d7785e5976ec049b46df5f1326af5a2ea6d103fd07c95385ffab0cacbc86'
```

Figure 3.1: Example usage of Hashing Library in Python

In this research, I compare the results using MD5, SHA1, and SHA256 hashing algorithms in an attempt to expose any performance differences due to hashing collisions.

3.1.2 Decision Trees

Decision Trees are a form of machine learning that are commonly used in data mining. As the name implies, decision trees can be visualized as a tree style structure, for large trees the human readable format can be large.

Wu *et al.* compared the performance of the top 10 algorithms for data mining, and specifically discussed some nuances of the C4.5 decision tree algorithm used in research [25]. In particular Wu notes that C4.5 differs from previous CART algorithms by supporting both binary and multi-way splits at each node [25] [20].

The C4.5 algorithm uses both information gain and gain ratio [25] with the objective of minimizing entropy with each node. The splitting process begins with a root node and continues recursively down to the final leaf nodes containing results. Thus the root node in a C4.5 decision tree always represents the largest separation in the dataset. Subsequent nodes down the tree become more and more specific to classifying a smaller percentage of the data.

For example, figure 3.2 shows an example decision tree for classification generated on a well known dataset using R (a project for statistics computing). Note how the top most node labeled as '1' is able to distinguish between 'setosa' and the other two types based on the 'petal length' feature being less than or equal to 1.9. In this example case, binary splits are being used, and the final classification for each data point read by the decision tree is shown by the largest column at the bottom of the graphic (representing the leaf). In the case that more than one class is present in a leaf graphic, those data points would be incorrectly classified by the model as the largest column in the leaf graphic.

To avoid overfitting (discussed later in section 4.3), pruning begins at the parents to leaf nodes further down the decision tree and the process continues up from the leafs to the root by accepting some predefined level of error in exchange for replacing some part of the decision tree with a simpler and more general version. Using the sample dataset example, figure 3.3 shows a pruned down and more general model compared to the earlier version shown in figure 3.2.

The freely available WEKA machine learning software [4] uses J48, the open

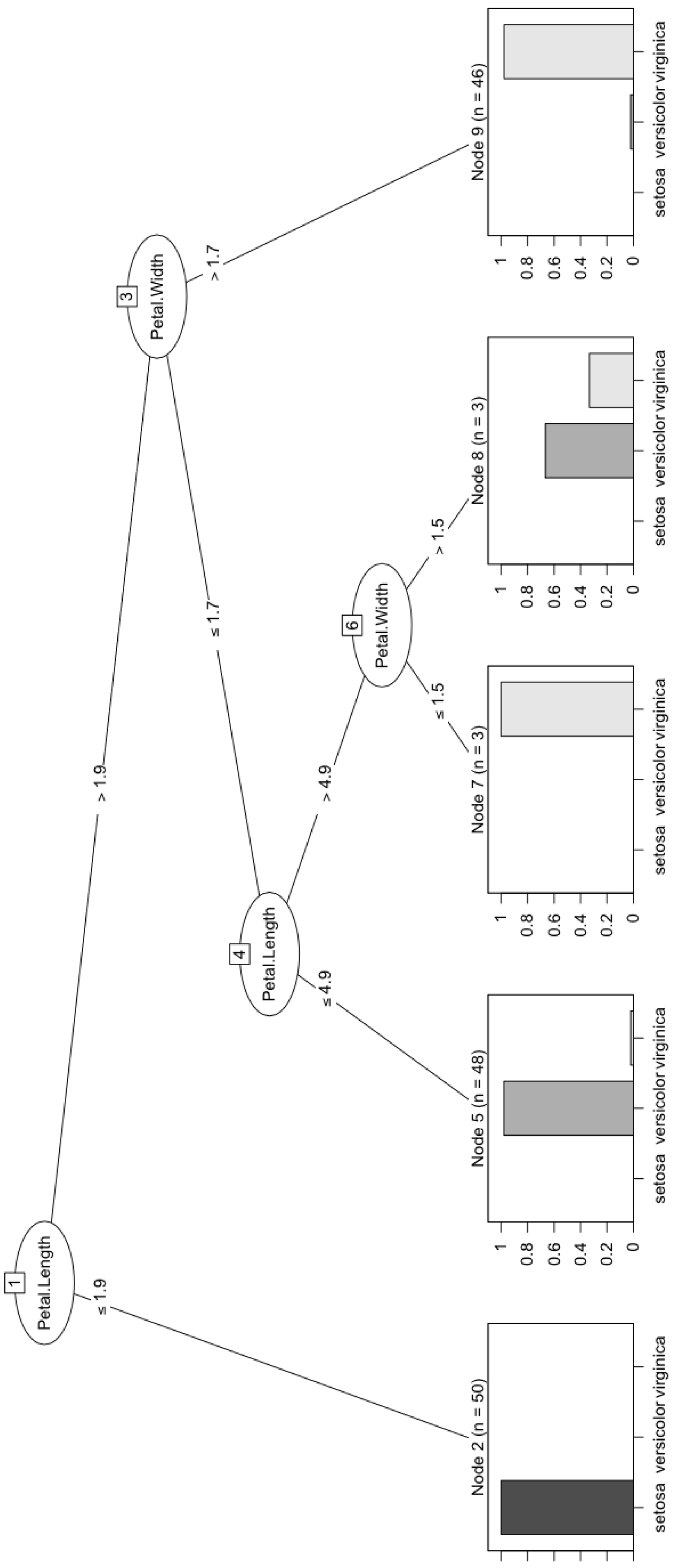


Figure 3.2: Example decision tree visualization for 1936 Iris (plant) dataset

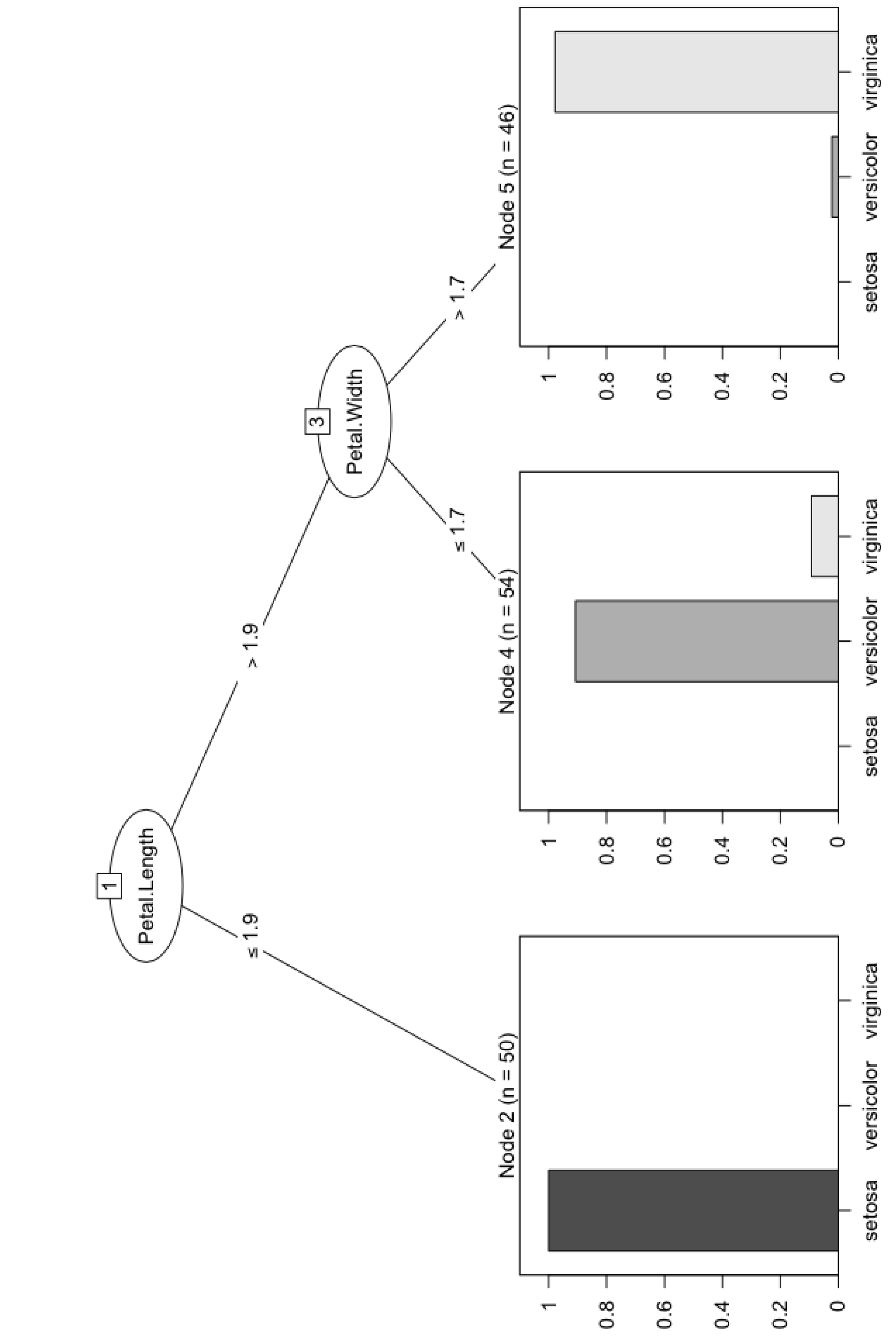


Figure 3.3: Example pruned decision tree visualization for 1936 Iris (plant) dataset

source implementation of C4.5 for decision trees. By default J48 allows multiway splits, pruning (subtree raising, collapseTree, confidence factor), and supports cross validation (discussed in section 4.3). 'R'[3], another freely available software perviously mentioned has several implementations of decision trees ranging from binary trees based on CART, and implementations based on newer algorithms such as C5.0. Matlab is also used throughout this research for verifying the results in the "fitctree" implementation of CART, and the interactive visualization tools for inspecting decision trees.

Decision trees have the advantage that they can be easy to visualize for simpler models. The C4.5 algorithm also provides a ruleset which allows easy inspection of rules per output class based on the logic of the unpruned decision tree.

Unlike other machine learning model types (such as SVMs, multi layer neural networks, and genetic programming), decision trees cannot synthesize new features to find separation between data on some higher dimensional plane (often called hyperplanes in SVMs [25]). The lack of feature synthesization in decision trees simplifies the underlying classification logic but can result in models that are larger than would be required when the data is separable when projected to a higher dimension.

3.2 Datasets

Three datasets were used during this research. The first two datasets referred to as GCU v1, and v2 from Glasgow Caledonian University [10] collected mobile data from 7, and 4 android devices respectively providing fine grain detail of processes running on the CPU as well as other hardware and software sensors. The third dataset referred to as RICE originated from Rice University LiveLab [22], and provides application level data for mobile data collected from 34 iPhone devices along with other hardware and software sensors.

3.2.1 GCU

GCU dataset version 1 is collected from 7 users consisting of staff and students of Glasgow Caledonian University. The data was collected in 2013 from Android devices.... The duration of the data varies from 2 weeks to 14 weeks for different users. Compared to other publicly available *datasets ..., it also contains a detailed diary for each user which allows for a detailed investigation of anomalies.* [9]

GCU dataset version 2 is collected from 4 users (specifically, 2 people, using their devices in two different periods covering different locations) consisting of staff and students of Glasgow Caledonian University. The data was collected in 2014 from Android devices ... The duration of the data is approximately 3 weeks. Only wifi networks are anonymized. [9]

Table 3.1: GCU mobile dataset features

Description	GCU V1	GCU V2
Rows	188,066	16,490,757
Unix timestamp	✓	✓
Available Probe Types	Cell, Wifi RunningApplications	Cell, Wifi RunningApplications Accel, Magnetic, Rotation Light, Noise, System
Users	7	4
Date (YYYY-MM-DD)	✓	✓
Time (hh:mm:ss)	✓	✓
Probe Values	One or more	One or more

Note: 'Date' will be omitted from future tables for clarity.

Both versions of the GCU datasets are provided in .TSV format which allows for simple processing with any basic scripting language. The original format can be described by the set of features as outlined in table 3.1. In order to compare the GCU v1 and v2 datasets, I will be restricting the features to 'Cell', 'Wifi' and 'RunningApplications' probe types which are common to both versions of the dataset.

Oddly, the different versions of the GCU datasets were anonymized differently. While open applications are each individually hashed using SHA224 for GCU V1, the applications names are human readable in GCU v2. WiFi names have been anonymized again, by individually hashing each Wifi network for GCU v1, where as GCU v2, shows the MAC [address] address rather than the SSID. Finally, the GCU v2 dataset provides two values for the cell probe which appear to be indexed, where GCU v1 only provides a single SHA224 hashed value.

Table 3.2: Original GCU Dataset Feature Types

Timestamp	Probe	User	Time	Values
1393833609	Cell	user2	00:00:09	Cell1, Cell2[1]
1387325587	Apps	user4	16:13:07	{Name, Name, Name, ...}
1387325654	Wifi	user1	16:14:14	{MAC, MAC, MAC, ...}

Note: 1. Cell2 feature is only available with GCU v2 dataset.

Referring to figure 3.4, note the different collection periods for the GCU dataset by randomly selecting two users. Hence going forward, I will only use time of day in seconds (written only as 'time' from this point forward), the 'year', 'month' and 'date' features will not be used because they would provide unrealistic amounts of entropy to distinguish between users.

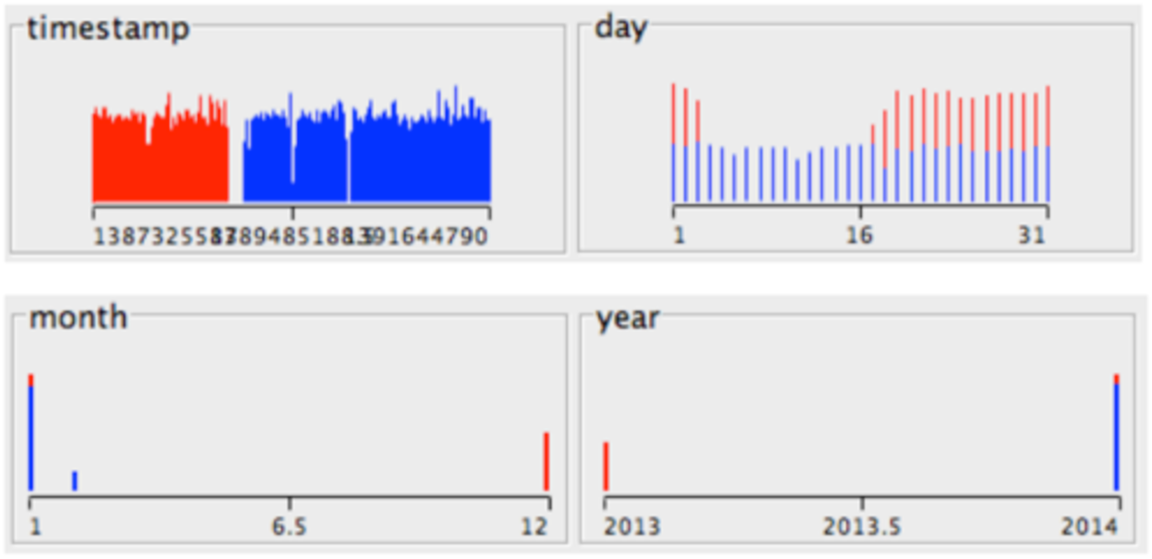


Figure 3.4: GCU timestamp and date features.

3.2.2 GCU Fine Grain Detail

One GCU probe in particular that provides an unexpected amount of detail in the logs is the Running Applications probe. In table 3.2 I showed a sample version of the values returned by the Running Application probe, but these values were formatted for demonstration purposes to fit in the table.

Table 3.3 shows the full list of values returned by a single sampling of the running applications probe. Among the list are popular apps such as '*Facebook*' and '*Whatsapp*'. Notably, unless the user was using some type of split screen view, the running application probe seems to be returning all processes running on the CPU, not just the current open application.

Another type of value shown in the detailed list that provides more fine grained detail is the group of applications related to widget services for news, email and weather. As the user decides to use more widgets on their screen, this will actually add background processes that the GCU 'Running Applications' probe will return in the log files.

Looking further at the values in table 3.3 notice more background service applications such as BluetoothHeadsetService, vevo.CastService, and Google VideoChatService. Presumably some of the services are running because the user; has enabled them; installed an application that requires them; or the device came configured to use these services out of the box.

The fine grain detail provided by the GCU running application probe may provide the foundation for fingerprinting users based on background services as well as actual open applications. The notion of device fingerprinting has been known for some time. Publicly available tools such as `amiunique.org`, `browserprint.info/` and `panopticlick.eff.org/tracker` allow for finger printing based on data exposed by web browsers. Beyond web browsers, research has shown device fingerprinting is possible based on active and passive approaches ranging from TCP clock skew [15], to wireless device drivers [8] just to name a few.

As with other forms of device fingerprinting, there is a considerable amount of information being leaked in the log files that users may not wish to have collected by third party applications ranging from application names themselves, to types of services, some of which can actually identify that a Samsung device is being used.

Table 3.3: GCU Running Application Probe

Running Application Values From Single Probe Sample
am.app.secretaudiorecorder.mainservice
ccc71.bmw.lib.bmw_service
com.android.exchange.security.ode.ODEService
com.android.internal.service.wallpaper.ImageWallpaper
com.android.phone.BluetoothHeadsetService
com.android.stk.StkAppService
com.android.systemui.statusbar.StatusBarService
com.facebook.analytics.service.AnalyticsService
com.facebook.fbservice.service.DefaultBlueService
com.facebook.push.c2dm.C2DMService
com.facebook.push.mqtt.MqttPushService
com.google.android.apps.youtube.core.transfer.DownloadService
com.google.android.backup.BackupTransportService
com.google.android.finsky.billing.iab.InAppBillingService
com.google.android.gm.provider.MailSyncAdapterService
com.google.android.gms.gcm.GcmService
com.google.android.gsf.gtalkservice.service.GTalkService
com.google.android.location.internal.server.GoogleLocationService
com.google.android.location.NetworkLocationService
com.google.android.videochat.VideoChatService
com.samsung.sec.android.application.csc.CscUpdateService
com.sec.android.app.FileTransferManager.FTSRunningChecker
com.sec.android.app.FileTransferServer.FTSService
com.sec.android.fotaclient.FOTAService
com.sec.android.inputmethod.axt9.AxT9IME
com.sec.android.providers.drm.OmaDrmConfigService
com.sec.android.socialhub.service.SocialHubService
com.sec.android.widgetapp.apnews.engine.WidgetService
com.sec.android.widgetapp.digitalclock.DigitalClockService
com.sec.android.widgetapp.emailwidget.EmailUpdateService
com.sec.android.widgetapp.stockclock.WidgetService
com.sec.android.widgetapp.weatherclock.WeatherService
com.seven.Z7.service.Z7Service
com.smart.monitor.appusage.AppService
com.smlds.smlStartService
com.vevo.cast.CastService
com.whatsapp.ExternalMediaManager
com.whatsapp.messaging.MessageService
com.wssyncmldm.DMService
edu.mit.media.funf.FunfManager
smsr.com.cw.TimeUpdateService

3.2.3 Rice Livelab

LiveLab is a methodology to measure real-world smartphone usage and wireless networks with a reprogrammable indevice logger designed for long-term user studies. We deployed LiveLab for a number of iPhone 3GS users. This includes 24 Rice University students from February 2010 to February 2011, and 10 Houston Community College students from September 2010 to February 2011. [2]

The Rice LiveLab dataset [2] is provided in a collection of SQL files, divided by feature as described in tables 3.4 and 3.5. In order to compare results from the GCU dataset, only a sub-set of the RICE datasets are examined for this research.

Table 3.4: Selected RICE dataset features to match GCU

Dataset	Rows	Trigger	Description
appusage.sql	740,672	user event	applications run by users
celltower.sql	321,318	periodic (15 minute)	cell tower connected
availablewifi.sql	654,540	periodic (15 minute)	available wifi access points

Table 3.5: Remaining RICE dataset features

Filename	Trigger	Description
apps.sql	-	list of all installed applications, among all users
call.sql	event	phone calls made received by users
sleep.sql	event	time phone spent in low power sleep mode
display.sql	event	display status
charging.sql	event	charging state of phone
power_detail.sql	event	built-in logfile driven)
accel.sql	periodic	accelerometer readings
iostat.sql	periodic	cpu and disk utilization)
cellsignal.sql	periodic	cell signal strength
loggeron.sql	event	time that the logger was running
associatedwifi.sql	periodic	connection to the associated WiFi access point
web.sql	event	web browsing history

Table 3.4 shows mobile apps, cell tower, and available wifi features that will be processed. As noted by the general dataset description of LiveLab [2], not all users

have data for all months. Since the ‘year’, ‘month’ , and ‘date’ features are not being used for this analysis of the GCU dataset, I will process the RICE dataset in a similar manner, only using the time of day in seconds (‘time’) feature so that I can compare the results more readily to the GCU results.

Some experiments were also conducted by re-introducing the day of the month feature along with ‘time’ (of day in seconds) to observe any performance differences for the RICE dataset. This will be clearly identified and discussed later in the results section.

Table 3.6: Example RICE mobile dataset features

Probe	User	Timestamp	Probe Values
			GeoId, TowerId
CellTower	A00	1269936873	6e2a5d5e0c0963ca11ab81e24d60f77a, dbb1c...
CellTower	A00	1269937616	6e2a5d5e0c0963ca11ab81e24d60f77a, dbb1c...
CellTower	A00	1269938533	6e2a5d5e0c0963ca11ab81e24d60f77a, ef437...
			SSID, BSSID, Channel, RSSI
Wifi	A00	1266245961	61480f2dd8b2ea4283ea321f915310b9, 3a7bf..., 11, -69
Wifi	A00	1266245961	f72c65b88838823c26e18332eb7a0278, b85d8..., 6, -70
Wifi	A00	1266245961	61480f2dd8b2ea4283ea321f915310b9, 2864b..., 6, -68
			Name, Duration
Apps	B04	1287334398	com.facebook.Facebook, 126
Apps	B04	1287334524	SpringBoard, 13
Apps	B04	1287334634	com.apple.MobileSMS, 68

Table 3.6 shows an example of the original RICE dataset features. CellTower, CellGeo and Wifi all use MD5 hashing to anonymize individual values such as human readable SSIDs. Similar to the GCU v2 dataset, the application names are provided in human readable format.

3.2.4 Comparison of Datasets

At a high level, both the GCU and RICE datasets contain information about Cell, Wifi, and Running Applications. Table 3.7 even notes that the ‘time of day’ can be derived from the timestamp, as well as ‘day of month’.

Table 3.7: Common Features Between Datasets

Dataset	Users	Rows	Timestamp	Time	Cell	Wifi	Running Apps
GCU v1	7	188,066	NA[3]	✓[6]	✓	✓	✓[1]
GCU v2	4	16,490,757	NA[3]	✓	✓	✓	✓[1]
RICE	34	1,716,530	NA[4]	✓[5]	✓	✓	✓[2]

Notes:	
1	GCU applications include main process, and sub processes.
2	RICE applications are limited to main application process only.
3	Timestamps are not applicable due to gaps in data collection between users in the same datasets.
4	'day' can be derived from the Timestamp and reintroduced to the RICE dataset to observe change in performance.
5	'time' (of day in seconds) can be derived from the Timestamp.
6	'Cell' feature for GCU v1 is provided as a single value instead of two separate values per probe tuple.

Referring to table 3.8, observe that both GCU and RICE have comparable representations of GeoId and TowerId comprising the 'Cell' feature (a single hashed column is provided in GCU v1), and SSID and BSSID represent the 'Wifi' feature. However, RICE returns a new tuple for each Wifi point collected at the same unix timestamp, GCU lists all the probe values together in a single tuple.

Table 3.8: RICE vs GCU Dataset Formats

Dataset	Probe	User	Timestamp	Probe Format
RICE	CellTower	string	unix	GeoId, TowerId
GCU V1	Cell	string	unix	Cell1
GCU V2	Cell	string	unix	Cell1, Cell2
RICE	Wifi	string	unix	SSID_{md5}, $\text{BSSID}_{\text{md5}}$, Channel, RSSI
GCU	Wifi	string	unix	{MAC, MAC, MAC, ...}
RICE	Apps	string	unix	Name, Duration
GCU	Apps	string	unix	{Name, Name, Name, ...}

Notes:	
	GCU Date and Time features have been omitted for clarity.
	Cell2 feature is only available with GCU v2.

To align the data formats of both datasets, I will convert the RICE data format to match the GCU data format as discussed in section 3.2.6.

Where the datasets diverge even further is in the data provided for running applications. The difference in the data collected may be attributed to the inherent security differences, and ability to gain root access to the device on Android vs iOS operating systems. GCU dataset was collected on the Android operating system and as noted in section 3.2.2, the probe returns a list of processes running (including background services). The RICE running applications probe (collected on iOS) only provides a tuple entry for each time the user changes primary application on screen (ie: one application at a time).

3.2.5 Transforming RICE Running Applications Data

Although I cannot accurately infer which background processes were running on the iPhone 3GS used in the RICE dataset (to better match the level of information in the GCU Running Application probe), I *can* build a running list of open applications similar to how applications would be listed in the multitasking view that was added in iOS 4.0.

The RICE dataset was collected on iPhone 3GS devices running a jailbroken version of iOS 3.0 at the time. Due to the more security conscious and locked down nature of the iOS operating system (compared to Android operating system the GCU datasets where collected on), researchers had to make use of a jailbroken version of the iOS operating system in order to gain root access required to run the scripts for logging device features [22].

The apple iOS 3.0 operating system did not support multitasking which would explain the single-application-at-a-time logging in the RICE dataset. However, partway through the Rice University study, iOS 4.0 was released and introduced the concept of multitasking which was supported by 3GS devices [1]. Furthermore, a jailbroken version of the iOS 4.1 operating system was available in July of 2011 [21]. The jailbroken version would provide the required root access to an iPhone which would support the installation requirements of the RICE livelab logging application [22].

Multitasking in iOS 4.0 (shown in figure 3.5) allowed the user to switch between apps by double clicking the home button which presented the user with a list of applications that the user had previously opened [1]. Similar to how appilcation launches where collected based on a user event (interupt) during the original RICE dataset collection [22], a double click system event would also have an interrupt that could be captured by a jailbroken device.

Figure 3.5: iOS 4.0 Multitasking [24]

The idea I am building off is a running list of open applications could be another way of creating a device fingerprint, one that is still available in iOS 10 as of this writing. Based on this idea, I can simulate the list of open applications from an earlier iOS version by keeping track of which applications a user has opened during the day. When an application name appears in the probe value indicating it was opened, the application name is added to the set of 'Running Applications', and when the date changes, the list is reset.

In the current version of iOS 10 (as of this writing), the multitasking application switcher will actually restore the list of past applications even after power cycling the device. Users must; purposefully swipe up to clear an application from the multitasking switcher; or do some form of reset on the device.

In implementing this list of running applications I will assume that users will not clear their open applications throughout the day from the multitasking view (as this has no performance impacts on iPhone devices). I will also assume a limited lifespan of apps in the multitasking list, that is, the list will be reset each night to prevent building up a permanently identifying digital fingerprint of each user over time. The second assumption may be relaxed on future datasets when the list of applications from the multitasking view is a recorded dataset feature, however, for this research, I did not want to bias the dataset by increasing the entropy for log entries near the end of the data collection period. A preview of the performance increase when relaxing the second assumption is provided in the Analysis section for reference.

3.2.6 Processing Datasets

Original datasets contain a tuple for each reading from each probe. To facilitate easier ingestion of datasets, the original data will be transposed into a running log where each probe type gets a feature column. In the transposed format, past values are shown on each row until they are updated or the date component of the input timestamp changes. This allows simpler correlation of values. The transposing preprocessing is accomplished using a python script which groups raw tuples based on the timestamp before outputting the data in the desired format.

Table 3.9: RICE vs GCU Dataset Format Wifi Example

Dataset	Probe	User	Timestamp	Probe Values
				SSID, BSSID, Channel, RSSI
RICE	Wifi	A00	1266245961	61480..., 3a7bf..., 11, -69
RICE	Wifi	A00	1266245961	f72c65..., b85d8..., 6, -70
RICE	Wifi	A00	1266245961	61480..., 2864b..., 6, -68
				{MAC, MAC, MAC,...}
GCU	Wifi	user1	1387325654	{27:bc:... , 35:0c:... , 6a:fd:.. ,...}

In the case of the RICE Wifi probe, multiple available wireless access points are returned as individual tuples with the same timestamp as shown in table 3.9. To align the data formats between datasets, a list of SSIDs is built when processing the RICE datasets and output as a single tuple containing all the SSID values. This will match the GCU format where a single Wifi probe tuple contains all the available MAC [address]s.

As discussed in section 3.2.5 the RICE 'Running Application' needs to be transformed from a single-application-at-a-time logging style, to keeping a list of open applications for each day. Again using python scripting to pre-process the original data, I transform the RICE 'Running Application' data into a running log by keeping a history of the applications opened by each user. To better match the stateless output of the GCU 'Running Application' probe, the history is stored in a sorted set of application names. The set is updated every time an application probe is processed for a particular user, and the sorted values are output as part of the transposed tuple as demonstrated in the **Original**, and **Transformed** sections of table 3.10.

Table 3.10: RICE vs GCU Dataset Format Running Applications Example

Dataset	Probe	User	Timestamp	Probe Values
				{Name, Name, Name,...}
GCU	Apps	user4	1387325587	{fb.service, fb.push, ggle.backup,...}
Original Format				Name, Duration
RICE	Apps	B04	1287334398	facebook, 126
RICE	Apps	B04	1287334524	SpringBoard, 13
RICE	Apps	B04	1287334634	SMS, 68
Transformed Format				{Name, Name, Name,...}
RICE	Apps	B04	1287334398	{facebook}
RICE	Apps	B04	1287334524	{facebook, SpringBoard}
RICE	Apps	B04	1287334634	{facebook, SpringBoard, SMS}

Walking through the data transposition in table 3.11 to table 3.12 , input tuples from the original dataset formats only update the relevant columns in the transposed dataset format. As more raw tuples are read, other columns populate with values for a given timestamp and the resulting row of feature columns output when the timestamp from the next raw tuple format increments to the next second. All feature value columns are then reset when the time gap between probe readings is over four hours for that user.

Table 3.11: Example original dataset tuples

Timestamp	Probe	User	Time	Probe Values
1387325187	GCU.Cell	user4	00:06:27	3, 254
1387325587	GCU.Apps	user4	00:13:07	{fb.service, fb.push, ...}
1387325654	GCU.Wifi	user4	00:14:14	{27:bc:..., 35:0c:..., ...}

Table 3.12: Example transposed dataset tuples to feature columns

Time (s)	Cell1	Cell2	Wifi	Apps	User
387	3	254			u4
46807	3	254		{fb.service, fb.push, ...}	u4
50414	3	254	{27:bc:..., 35:0c:..., ...}	{fb.service, fb.push, ...}	u4

One of the results of transposing the datasets is reducing the number of rows as listed in table 3.13 (and subsequent dataset file size) due to the fact that rows store values for multiple probes.

Table 3.13: Reduction in Row Counts After Transposing

Dataset	Users	Original Rows	Transposed Rows
GCU v1	7	188,066	124,154
GCU v2	4	16,490,757	3,392,454
RICE	34	1,716,530	1,113,847

3.3 Baseline Method

To establish a baseline performance on the datasets without any additional anonymization, the datasets are processed to output a simpler transposed format.

Table 3.14 shows one example of transposing three probe readings. Each distinct feature value is assigned a feature column, boolean Y/N indicates the presence /

absence. This baseline is obviously quite crude, but serves to provide a basic reference point for classification model performance when no additionalanonymizationis applied. Concerns regarding overfitting are discussed in section 4.3.

Table 3.14: Example Transposing to Feature Column per Distinct Value

Original Dataset Tuples

Timestamp	Probe	User	Time	Probe Values
1387325187	GCU.Cell	user4	00:06:27	3, 254
1387325587	GCU.Apps	user4	00:13:07	{fb.service, fb.push, ...}
1387325654	GCU.Wifi	user4	00:14:14	{27:bc:..., 35:0c:..., ...}

Transposed Dataset Tuples for Baseline

Time (s)	Cell1_3	Cell1_4	Cell1_...	Cell2_254	Cell2_172...	Cell2_...	Wifi_27:bc:...	Wifi_35:0c:...	Wifi_...	Apps_fb.service	Apps_fb.push	Apps_ggle.backup	Apps_...	User
387	Y	N	...	Y	N	...	N	N	...	N	N	N	...	u4
46807	Y	N	...	Y	N	...	N	N	...	Y	Y	Y	...	u4
50414	Y	N	...	Y	N	...	Y	Y	...	Y	Y	Y	...	u4

Due to the size of the datasets used, and the number of feature columns generated from the number of distinct values for certain features, the baseline method does not scale particularly well. Note in table 3.15 that the 'Wifi' feature has several orders of magnitude more distinct feature values than the other probe types.

Table 3.15: Distinct Number of Feature Values by Dataset

Dataset	Users	Cell1	Cell2	Wifi	Running Apps
GCU v1	7	2375	-	**14095**	461
GCU v2	4	76	1550	**25478**	252
RICE	34	6849	467	**15029**	2301

Chapter 5 will detail the results of the baseline experiments, however, due to the sheer number of distinct wifi feature values and corresponding memory requirements for building a classification model, no baseline method could be completed when the

'Wifi' feature was included. In the event that the baseline methods with 'time' and 'Apps', as well as 'time', 'Cell' and 'Apps' achieve greater than 90% accuracy, I will assume that adding additional features would either boost classification performance, or have no effect at all.

3.4 Proposed Anonymization Method

Using three datasets from two different sources summarized in table 3.7 and 3.8, each with varying levels of pre-existing anonymization, features comprised of a set of nominal values will be hashed to anonymize the log files.

A list of one or more; cell towers; available Wifi access points; and open applications are all nominal values that will appear identically on each mobile device. For this research, it is assumed that the user does not wish to expose individual names, or even how many of each attribute are present in the log files for their device.

By concatenating the list of one or more values for each of the respective nominal features and hashing the concatenated value, I am both obscuring the concatenated values, and obscuring the number of distinct values that were recorded for a given sensor. Note, this method is applicable even if the original datasets have already anonymized individual nominal names, such as was done with the WIFI SSID featue.

The hash H_X is generated by concatenating a list of one or more values from feature X, where X can be any nominal value which is consistent across mobile devices.

$$H_X = hashFunction(X_1|X_2|...|X_n) \tag{3.2}$$

The operation of converting a list of values for a single feature data point can be viewed as a form of global recoding [26] since the hash function will remain constant across all users in this research.

For example: the proposed transpose and hashing would process the original 'WiFi' probe from either the GCU or RICE datasets as shown in figure 3.6, using the subscript $_W$ to denote the 'Wifi' probe.

To condense the datasets and models as well as speed up classification the hexadecimal hash can also be indexed per feature column as shown after the hashing step in figure 3.6.

Raw tuple:

Timestamp	Probe	User	Time	Probe Values
1393833615	WifiProbe	user2	00:00:15	{ 27:bc:4d:b5:09:af 35:0c:b5:8a:71:77 6a:fd:c7:c8:1c:aa }

H_W = hashFunction(27:bc:4d:b5:09:af35:0c:b5:8a:71:776a:fd:c7:c8:1c:aa)
i_{H_W} = indexHash('wifi', H_W)

Transposed tuple:

time	cell1	cell2	wifi	apps	user
15			i_{H_W}		u2

Figure 3.6: Example transposing and hashing original GCU tuple to feature columns

Using machine learning model performance for user classification on the transposed and hashed data features, an attempt to identify individuals will be the measure of information leakage (k-anonymity [23]) and device fingerprinting in this research.

The concatenation and hashing anonymization method I am proposing increases the input search space for a given hashing algorithm. For example, if a malicious user obtained some device logs, they might attempt to hash a popular application name such as "Facebook" using MD5. Obtaining the hash ''d85544fce402c7a2a96a48078edaf203' the attacker can check to see if it matches any of the hashed entries in a log file, however, they find none.

To explain why the attacker would now have to expend exponentially more time to match entires in the log file, here is an example of where the 'Facebook' application might appear in the log files and how it would be handled. Similar to figure 3.6, 'Facebook' is only ever listed together with other applications, for the purpose of our example, the shortest concatenated value that 'Facbook' appears in is 'FacebookSafariSMSYahooWeather'. The value stored in the log files would then be the hash of 'FacebookSafariSMSYahooWeather', which (again using MD5) would be '309d2c2ca5e4a79eb4ead142046fe84e'. The one-way nature of hashing means that there is no way to compare two hashes for similarity of their input values.

In section3.1.1 I used passwords as an example for hashing. Quite obviously if all passwords were only one character, or always the same number of characters, they would not be very secure. Unlike passwords that have a limited number of possible characters per position (26 lower case letters, 26 upper case letters, 10 numbers, and approximately 32 symbols), each application name that can be concatenated for the input to a hashing function can be composed of multiple characters. Even using the top N popular applications, an attacker would have to expend N^2 time to generate values and compare against log files generated by concatenating and hashing.

Chapter 4

Dataset Challenges

This chapter discusses some of the challenges encountered processing different datasets, ranging from variability of features and collection techniques, to dataset local. The effects and steps to mitigate overfitting are also discussed as they pertain to decision trees.

4.1 Variation In Source Dataset Features

As discussed in section 3.2.4, the GCU and RICE datasets have collected different types of application data using different methods. The GCU datasets collected processes running on the device at periodic intervals where as the RICE dataset collected application names based on user events. In Section 3.2.5 I detailed the method used in this research to attempt to match the level of entropy provided with the GCU dataset based on some reasonable assumptions about how the iOS on the iPhone worked at the time and continues to operate today.

However, there are two main limitations to my approach. First, the GCU 'Running Applications' probe provides details (see section 3.2.2) of additional background services, and screen widget configurations which are not provided by the RICE dataset (or iOS in general until several years later in iOS 10). Second are the limitations imposed by my assumptions regarding user behaviour with the multitasking list of open applications on an iOS device. I noted in section 3.2.5, that the list of open applications is reset at each date change, and also assume that users would not otherwise clear applications from the multitasking apps list.

Specifically regarding the assumed lifetime of open applications in the simulated multitasking apps list, an alternate approach could have used. This alternate approach might be more inline with one type of iOS device user (such as the author of this research), that is, a user that randomly closes some average percentage (for example 30%-60%) of apps in the multitasking view once ever 3-5 days. There are surely other types of users for iOS devices, some no doubt routinely keep their multitasking view completely empty. Without adequate statistics relating to user tendencies on iOS devices, I proceeded on the assumption that my initial ‘daily-reset’ of open apps was a reasonable compromise within the scope of this research.

4.2 Location vs. User Based Features

When the baseline method was detailed in section 3.3, table 3.15 showed evidence that both location based (‘Cell’ and ‘Wifi’) and user based (‘Apps’) features increased in the number of distinct probe values as the number of users increases across datasets.

Given that there are simply a greater number of Wifi access points in most areas than there are cellular towers, I would have presumed that the number of distinct ‘Wifi’ probe values would have been the most sensitive to user count. For example, walking down a single different street on the way home from work or university would pick up numerous different Wifi access points, while still remaining in the same one or two cellular areas.

The difference between the volume of distinct ‘Wifi’ probe values between the two GCU datasets can be attributed to data collection interval differences that unfortunately are not detailed with the dataset.

Table 4.1: Distinct values by RICE probe vs. number of users.

Number of Users	CellGeo	Apps	CellTower	Wifi
4	106	389	1173	2704
7	156	680	2076	2989
14	221	1233	3213	6728
25	345	1847	5196	11415
34	442	2236	6576	14700

However, taking a closer look at the RICE dataset, figure 4.1 shows a better representation of the correlation between the number of distinct probe values and the number of RICE users. With a sample size of 34 users, smaller sample sizes can be randomly selected to observe the effects of increasing the number of users on the number of distinct probe values.

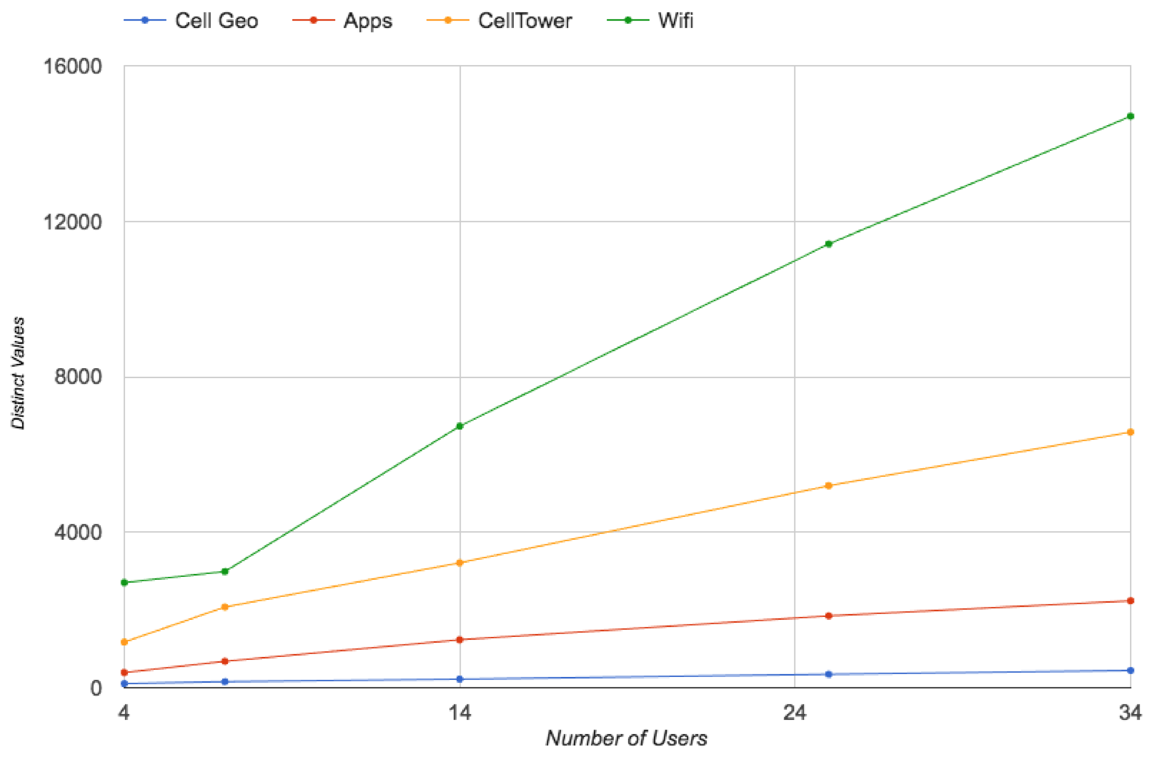


Figure 4.1: Distinct values by RICE probe vs. number of users.

Note, distinct feature values reported in table 4.1 are the average of random sampling of 4 users 100 times, and random sampling 7-34 users 20 times each to account for any bias introduced by individual users.

After randomly sampling users from the RICE dataset, my presumption about the 'Wifi' feature being the most sensitive to number of users would appear to be confirmed. Not only does the 'Wifi' feature have the largest number of distinct values, it also increases the most as users are added.

Shifting the focus to the 'Cell' feature, there is a proportional increase in the number of distinct probe values as the number of users changes from one dataset to another. This makes sense, although users were co-located according to their dataset, the collection area was not limited to a laboratory. As different users live in different parts of a geographic area surrounding the dataset origin, I would expect to see a steady increase in the number of distinct probe values related to cellular features. At some number of users N, the number of distinct cell features would start to plateau within the context of a user study as the number of users willing to participate in a study will begin to become geographically limited unless the study was conducted online, and participants were randomly selected irrespective of their geographic location.

The distinct number of ‘Running Applications’ (labeled as ‘Apps’) also shows an increase in relation to the number of users in each dataset. However, the rate of increase after 14 users drops more considerably than ‘Wifi’ or ‘Cell Tower’ features.

4.3 Overfitting

In the domain of machine learning, overfitting occurs when the classification model (in this research) has been optimized to classify only the training data. Models that overfit their training data often perform well in the training results, for example, 90% accuracy, but then fall suddenly to a much lower level of performance, for example, 40% when the model is used to classify a different set of previously unseen data.

There are several ways to detect and mitigate against overfitting. The main problem with overfit models is that they are optimized for a specific set of data, this usually results in larger model sizes, both in terms of complexity, but also in terms of raw size compared to the dataset. For example, if the dataset is 10mb in TSV format, and the trained model is 150mb when saved to disk, it could be an indication that the model is overfitting the training data.

The first way to mitigate is simply partitioning the data into training, validation, and test sets. The model is built based on the training data, and then after a set number of iterations, the model is validated on the validation dataset. The difference with the validation set, is the process of building the model does not receive feedback about the accuracy of prediction on the validation data. This continues while the validation set continues to increase in performance and halts when the model starts to become overfit and thus lowers the performance on the validation dataset. The final evaluation of the model is then determined by using the model to evaluate the test dataset which has not been used up to this point. Using the training / validation / test dataset partitioning, the data is randomly assigned two one of the partitions, training data usually comprises 66% to 75% of the dataset, and the remaining 33% to 25% being roughly split into two thirds validation data, and 1/3 test data.

A slight improvement on the random split of data to generate a model, and subsequently the method used in this research is k-fold cross validation [14]. In this approach, the dataset is randomly assigned to k buckets, where 90% is used for training, and the final 10% is used for testing. A different 10% of the buckets is used for testing each of the following k models until each model has been validated on a different partition of the data. The final evaluation is the average of the k models. For this research, 10-fold cross validation is used as implemented in Weka machine learning software [4] when presenting model accuracy.

Decision Tree's in particular have a number of methods for mitigating overfitting. Limiting parameters such as tree depth; leaf count; and the minimum number of instances to create a leaf node are all easily tuneable. Another parameter is adjusting the pruning confidence which attempts to prune off branches of the tree which do not increase the evaluation of the model based on confidence factor. Using the Weka machine learning software in this research, pruning is enabled by default and left on with a default confidence factor of 0.25, and only the minimum number of instances to create a leaf node is varied along a logarithmic scale. In Matlab, the minNumLeafs argument was varied along a log scale with the subsequent model performance recorded.

Figure 4.2 shows the relationship between GCU model size, and the effect of increasing the minimum number of leafs required to spawn a new branch. Another example of pruning decision trees can be done by depth. Matlab provides a visualization tool that allows pruning directly in a decision tree visualization. Figure 4.3 shows decision nodes (shown in grey dashed lines) being replaced by leaf nodes with the class that yields the lowest error if the node must be pruned. This pruning is an example of how the decision tree would change if the minimum number of leafs value was increased, or if another measure such as "maximum tree depth" was used with a value of 4. The deeper parts of the decision tree would be pruned away in an attempt to reduce the model size, and limit overfitting.

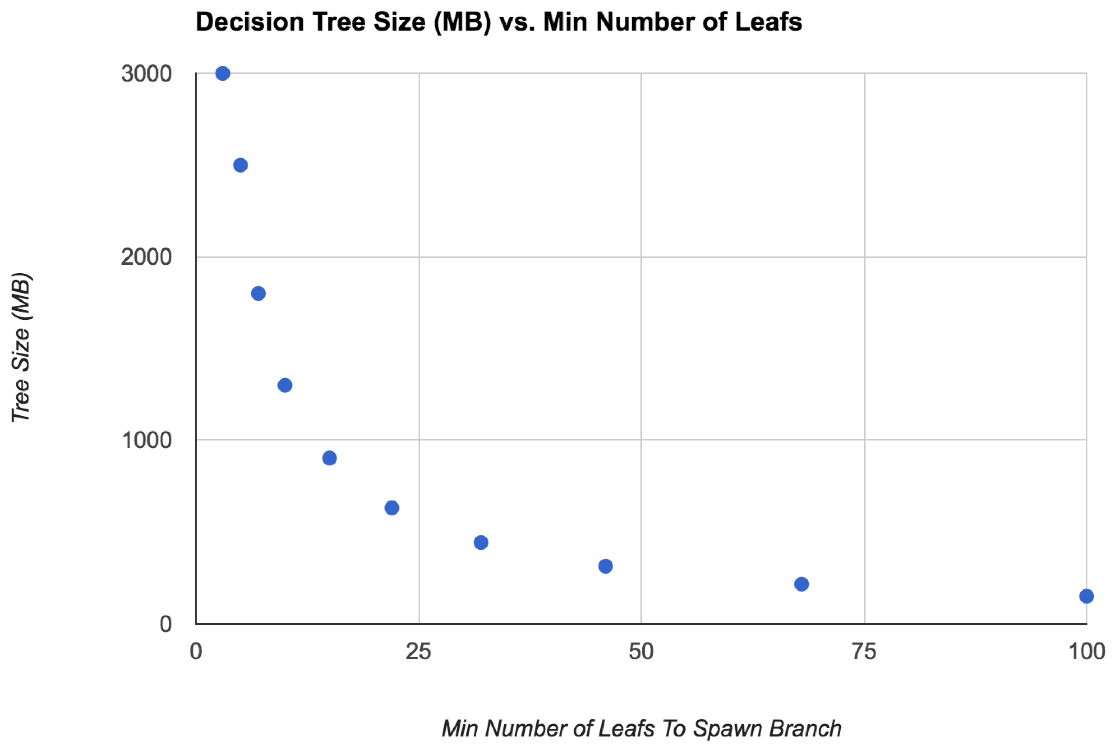


Figure 4.2: GCU v2 Decision Tree Size (MB) vs. Min Number of Leafs

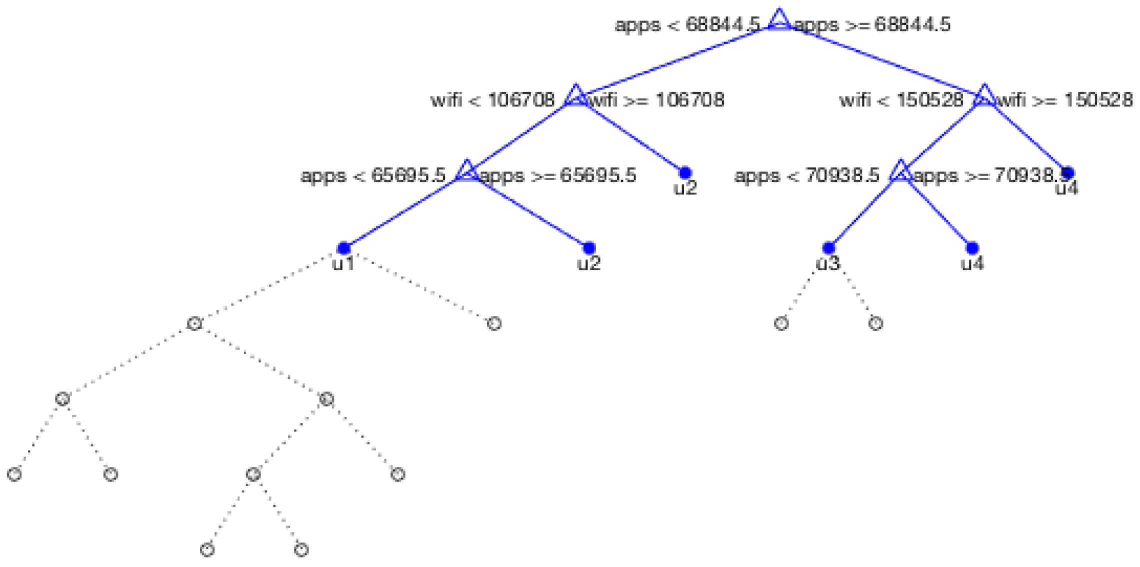


Figure 4.3: Example pruned decision tree

Chapter 5

Results and Evaluations

Results for both datasets are reviewed in detail, including sampling the RICE dataset for different number of users. Results on GCU and RICE datasets are then compared and discussed. Values presented are from the WEKA C4.5 decision three classifier implemented in J48, and verified on Matlab using the “fitctree” implementation of CART.

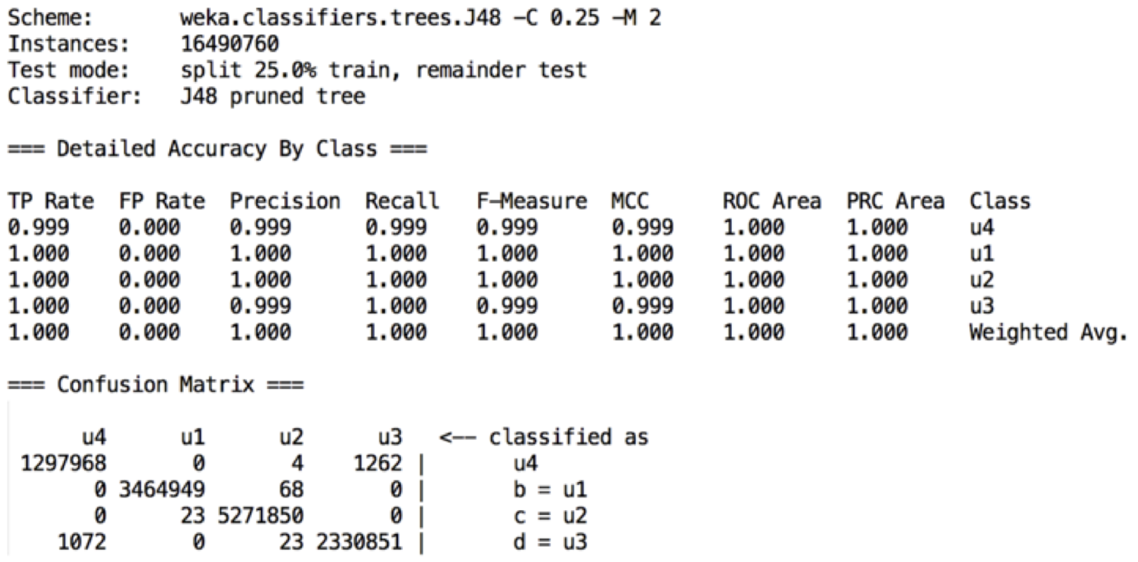
5.1 GCU Results

Using all the anonymized features (‘Cell’, ‘Wifi’, and ‘Running Applications’) which have been hashed and indexed to a continuous integer range, the GCU mobile datasets are nearly perfectly separable by user label. The confusion matrix from one experiment in figure 5.1 shows %99.99 for accuracy, precision, and recall. These results are further confirmed using 10 fold cross validation, on all three of the hashing algorithms (MD5, SHA1, SHA256) and by varying which features are used to build the classification model as detailed in table 5.1.

Table 5.1: GCU Classification Model Results

Dataset	Users	Time	Apps	Cell	Wifi	Baseline	MD5	SHA1	SHA256
GCU V1	7	✓	✓			99%	99%	99%	99%
GCU V1	7	✓	✓	✓		99%	99%	99%	99%
GCU V1	7	✓	✓	✓	✓	×	99%	99%	99%
GCU V2	4	✓	✓			99%	99%	99%	99%
GCU V2	4	✓	✓	✓		99%	99%	99%	99%
GCU V2	4	✓	✓	✓	✓	×	99%	99%	99%

Notes 1 All hashes are indexed after hashing, per section 3.4
2 All results reported are based on 10-fold cross validated accuracy.
3 × denotes out of memory due to number of wifi instances.



```
Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Instances:    16490760
Test mode:    split 25.0% train, remainder test
Classifier:   J48 pruned tree

=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
0.999    0.000    0.999      0.999    0.999      0.999    1.000     1.000     u4
1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     u1
1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     u2
1.000    0.000    0.999      1.000    0.999      0.999    1.000     1.000     u3
1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Weighted Avg.

=== Confusion Matrix ===

      u4      u1      u2      u3   <-- classified as
 1297968       0       4    1262 |       u4
       0 3464949      68       0 |       b = u1
       0      23 5271850       0 |       c = u2
    1072       0      23 2330851 |       d = u3
```

Figure 5.1: GCU decision tree confusion matrix

Using Matlab to visualize some of the feature combinations, models are inspected visually for size, depth, or other attributes that may be of interest. Figure 5.2 shows a classification tree that has been generated during the 10-fold cross validation process. The model stays equally compact for the ‘Cell’ - ‘Apps’ and ‘Wifi’ - ‘Apps’ feature combinations.

When only the ‘Time’ and ‘Running Applications’ features are used to classify the GCU v2 dataset, the model does increase in size. The model visualized in figure 5.3 appears much more bushy and deep. However, given that the model is classifying 3,392,454 data values, this model is acceptable with a cross validation performance of 99% accuracy.

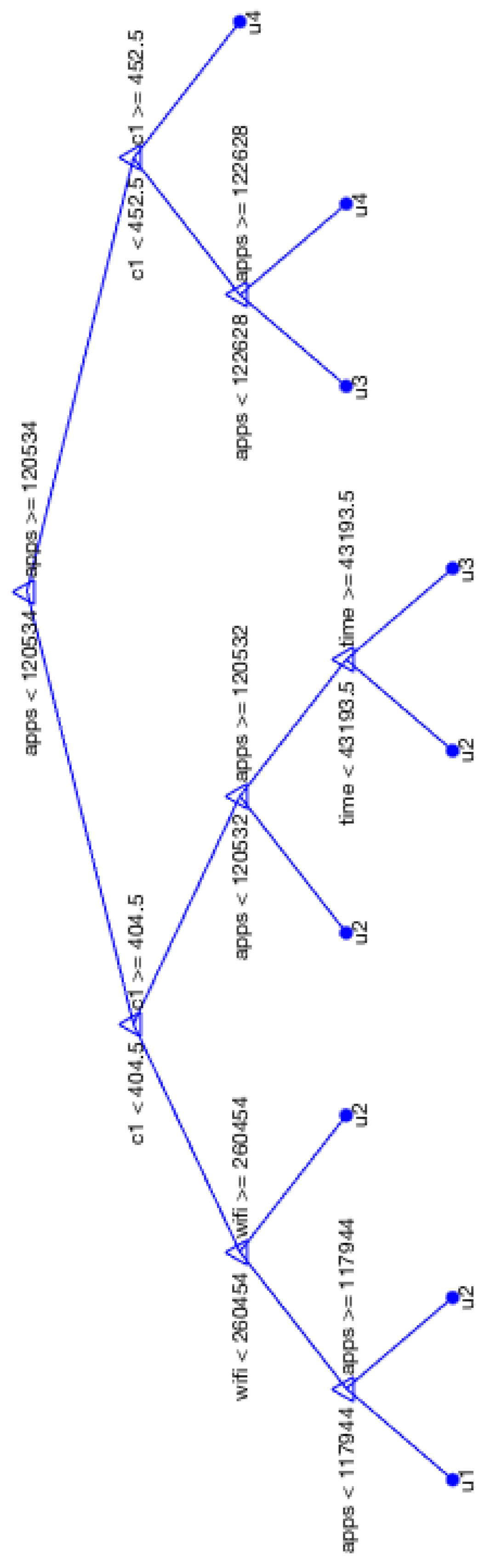


Figure 5.2: GCU v2 classification model based on all available features

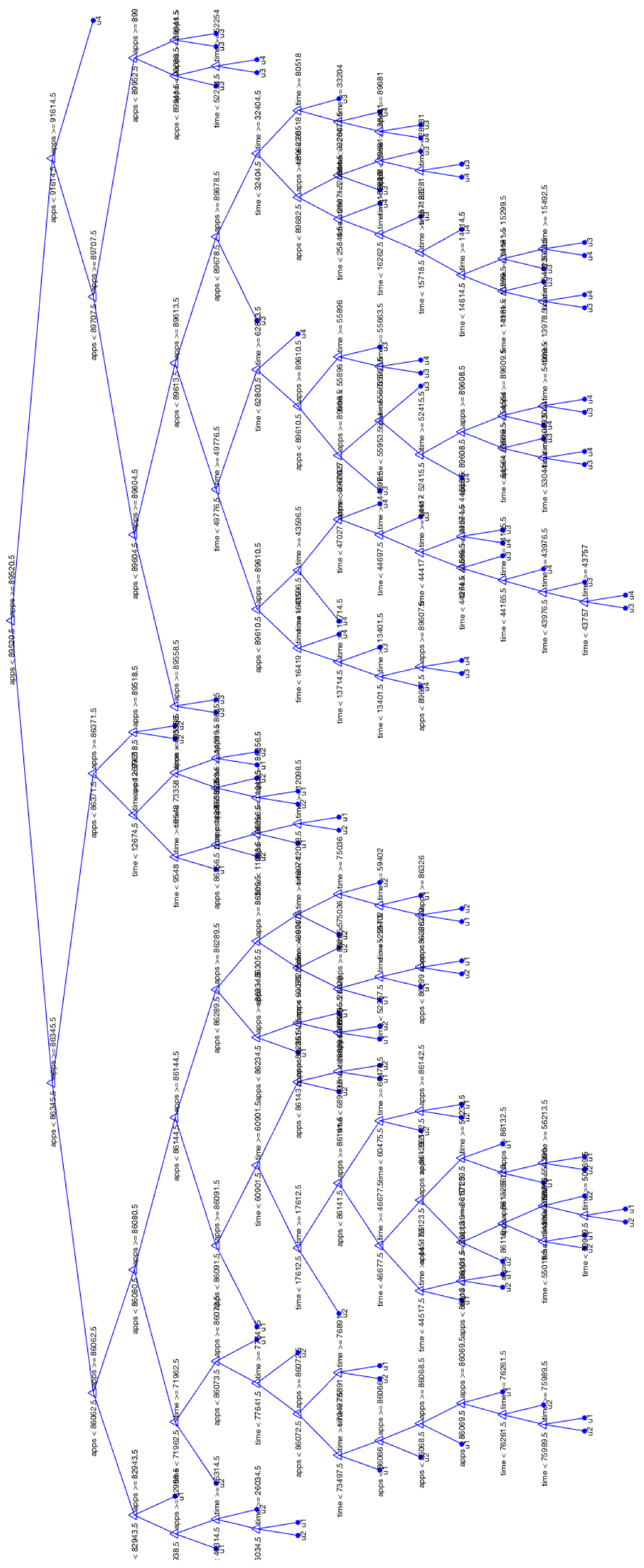


Figure 5.3: GCU v2 classification model based on time and app features only

5.2 RICE Results

The classification performance of the RICE dataset varies based on the features used to build the model. Using all the anonymized features ('Cell', 'Wifi', and 'Running Applications') which have been hashed and indexed to a continuous integer range, the RICE dataset is 13-66% separable by user label with 3% being the level of accuracy a random choice machine would achieve on average. Reviewing table 5.2, the largest performance decrease occurs when the 'Cell' feature is not part of the set of features used to build the model. Reintroducing the 'Day' feature (integer day of the month) increases classification accuracy by 2-3% in all feature combinations.

Table 5.2: RICE Classification Model Results

Users	Time	Day	Apps	Cell	Wifi	Baseline	MD5	SHA1	SHA25
34	✓		✓			98.50%	13%	13%	13%
34	✓		✓		✓	✕	18%	18%	18%
34	✓		✓	✓		99%	62%	62%	62%
34	✓		✓	✓	✓	✕	65%	65%	65%
34	✓	✓[4]	✓			99.50%	16%	16%	16%
34	✓	✓[4]	✓		✓	✕	20%	20%	20%
34	✓	✓[4]	✓	✓		99%	66%	66%	66%
34	✓	✓[4]	✓	✓	✓	✕	68%	68%	68%

Notes 1 All hashes are indexed after hashing, per section 3.4
2 All results reported are based on 10-fold cross validated accuracy.
3 ✕ denotes out of memory due to number of wifi instances.
4 'Day' refers to integer day of the month.

One item to note in table 5.2 are the identical results of each of the hashing algorithms. This would seem to indicating either hashing collisions are not a concern, or all of the hashing collisions are equally being impacted by hashing collisions yielding the same results. The probability of the later would be extremely unlikely, therefore I will proceed under the assumption that results will be the same regardless of hashing algorithm used. For a full listing of results, please referrer to appendix C.

In order to compare the results from the RICE dataset directly to the GCU datasets, classification models are also built by again randomly sampling 4 & 7 users from the RICE dataset. Furthermore, controlling for the number of users allows for observing the gradual effects of the number of users on classification accuracy for the RICE dataset.

Random sampling of users was done three times per user sample size, the results of the cross validation classification accuracy were averaged from the three different user samplings.

Number of Users	Apps	(+Day) Apps	Apps Cell	(+Day) Apps Cell	Apps Wifi	(+Day) Apps Wifi	Apps Cell Wifi	(+Day) Apps Cell Wifi
4	51%	53%	91%	94%	56%	55%	92%	95%
7	37%	41%	77%	85%	38%	44%	77%	86%
14	24%	26%	74%	78%	27%	28%	75%	79%
25	17%	20%	66%	70%	22%	24%	69%	72%
34	13%	16%	62%	66%	18%	20%	65%	68%

Table 5.3: Rice Classification Accuracy vs. Number of Users

The results of randomly sampling users shown in table 5.3 and figure 5.4 show a steady correlation of decreasing classification performance as more users are sampled to build and validate the model. The accuracy results for 4 randomly selected users (matching the number of users for the GCU V2 dataset) ranges from 51 to 95%. For 7 randomly selected users (for comparison to the GCU v1 dataset) ranges from 37% to 86%. The higher accuracy scores are still the result of the inclusion of the 'Cell' feature which has a minimum +35% effect on classification accuracy regardless of the number of users.

Figure 5.5 takes a closer look at the effects of increasing the number of users on the size of the decision tree model. Here size is based on the total number of nodes (full details available in Appendix C table C.4). The models that include the 'Cell' feature are consistently 50% of the size of models without when randomly sampling 14 or more users. The classification accuracy of these smaller models is also higher as previously noted in table 5.3 and figure 5.4. Also note, the size of the the model increases proportional to the size of the dataset (shown in green and based on the number of users). One final observation I will draw from figure 5.5 would be the increase in model size with the addition of the 'Day (of the month)' feature.

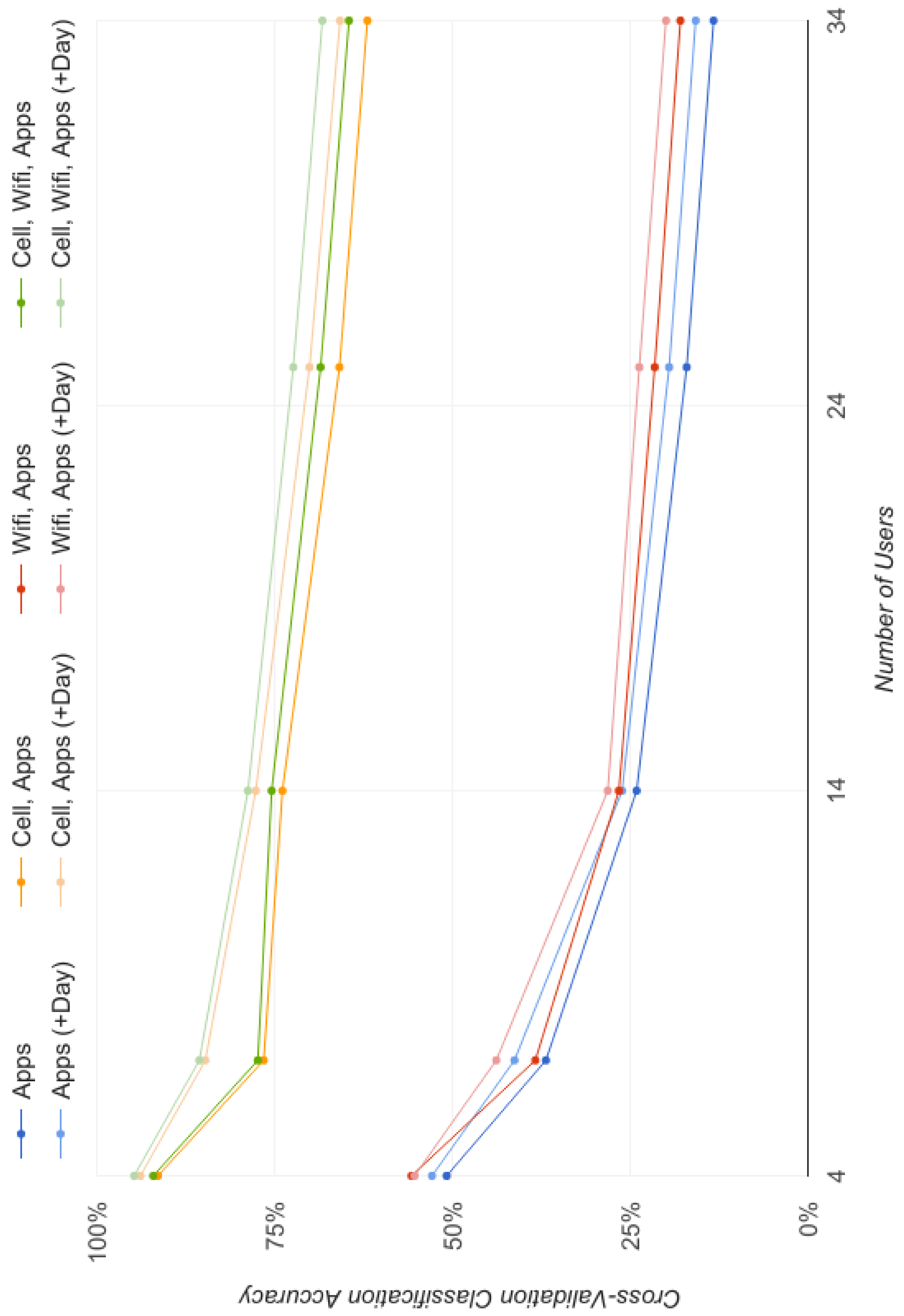


Figure 5.4: Rice Classification Accuracy vs. Number of Users

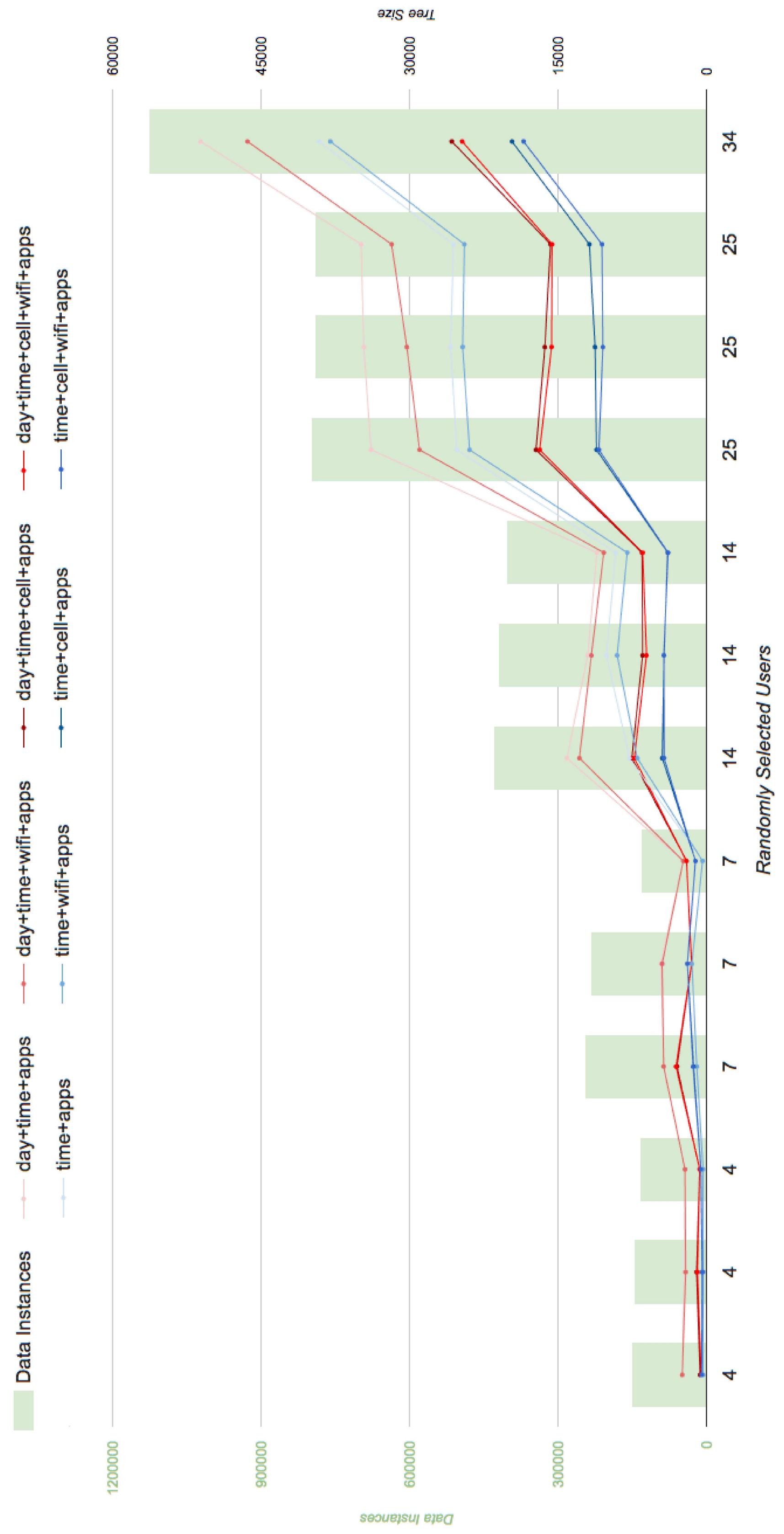


Figure 5.5: Rice Tree Size vs. Number of Users

5.3 Analysis

The results for the GCU (section 5.1) and RICE (section 5.2) datasets show different levels of classification accuracy. By randomly sampling four and seven users from the RICE dataset, I attempted to obtain results that were as comparable as possible to the GCU dataset.

Dataset	Number of Users	Time	Apps	Apps Cell	Apps Wifi	Apps Cell Wifi
GCU V1	7	✓	99%	99%	99%	99%
GCU V2	4	✓	99%	99%	99%	99%
Rice	4	✓	51%	91%	56%	92%
Rice	7	✓	37%	77%	38%	77%
Rice	14	✓	24%	74%	27%	75%
Rice	25	✓	17%	66%	22%	69%
Rice	34	✓	13%	62%	18%	65%

Gray identifies results below 60% classification accuracy.

Table 5.4: Comparison of Classification Accuracy vs. Number of Users

The GCU datasets were limited to 7 users for V1, and 4 users for V2, however both showed 99% classification accuracy when only using the 'Time', and 'Apps' features. The results from the same number of randomly selected users from the RICE dataset showed 48% and 62% lower performance as outlined in the 'Apps' column of table 5.4. However, adding the 'Cell' feature does increase the classification accuracy for the RICE datasets to 91% for 4 users, and 77% for 7 users. Adding the 'Wifi' feature provides a modest increase of 1% to 5% in classification accuracy.

The largest remaining difference between the datasets, even when comparing results for the same number of users is the different levels of entropy contained in the 'Running Applications' probe. As discussed in section 3.2.2 and 3.2.5, the GCU datasets provide a list of running processes at a given point in time. The collection of such a level of detail provides enough entropy that not only can individual application names be anonymized, but the entire list of applications can be anonymized, hiding details such as the number of open applications. The proposed method of concatenation and hashing greatly increases the difficulty of dictionary and brute force attacks that can be carried using a list of popular applications from apps stores.

In contrast the RICE 'Running Applications' probe is limited to a single application name (and duration of usage which was not used in this research). My attempt at re-constructing a list of applications that would be shown in the iOS multi-tasking view does not appear to provide the same level of entropy as the GCU 'Running Applications' values. This could be in part due to the nightly reset of multi-tasking list of applications. Varying the reset time of the multi-tasking list of applications for the RICE datasets should increase the classification performance, however, the reset interval is highly user specific statistic. As of this writing the list of applications in the multi-tasking view is still user controllable, requiring users to swipe up to individually clear each application from their open history. Some users may leave the list of applications in the multi-tasking view untouched most of the time, while at the others may swipe to clear applications immediately after use.

Going to the extreme of never reseting the list of open applications a user has used on their mobile device, table 5.5 shows the increase in performance if a user never cleared applications from the multi-tasking view.

Table 5.5: RICE Classification Model With No Application List Resets

Users	Time	Apps	Cell	Wifi	Baseline	MD5	SHA1	SHA256
34	✓	✓			98.50%	98.7%	98.7%	98.7%
34	✓	✓		✓	×	98.7%	98.7%	98.7%
34	✓	✓	✓		99%	99.4%	99.4%	99.4%
34	✓	✓	✓	✓	×	99.4%	99.4%	99.4%

Notes	1 All hashes are indexed after hashing, per section 3.4
	2 All results reported are based on 10-fold cross validated accuracy.
	3 × denotes out of memory due to number of wifi instances.

Even with the differences in source data for the 'Running Applications' probe, the results for the RICE dataset are still able to achieve over 60% classification accuracy when the cell probe is used for up to 34 users using the proposed concatenation and hashing anonymization method.

Chapter 6

Conclusion And Future Work

So how does information leakage change when varying the level of anonymization in dataset feature values? Using three datasets from two different datasources I attempted to answer this question.

Initially comparing encoding and encryption, I went on to discuss how hashing algorithms are used in one way functions and the implications of using them for log files. I defined information leakage as the user classification accuracy of log files using decision tree implementations from Weka(C4.5), Matlab(CART), and R(C5.0).

I focused on going beyond hashing individual values. Although I made use of encoding by indexing values to prepare them for ingestion to machine learning software packages, this was only done to reduce the storage size of decision tree models. The underlying difference in my proposed method is the combination of nominal values, plain text or already encrypted, into one concatenated value that forms the input for a hash function. Throughout this research I showed that my results were comparable using either MD5, SHA1 or SHA256 hashing algorithms.

The GCU datasets with 7 (V1) and 4 (V2) users showed that even when information (consisting of several distinct values) is condensed in hash form, users can still be distinguished with greater than %99 accuracy, precision and recall. These findings indicate that there are ways of completely obfuscating the name and number of, applications and Wifi access points using a single hash value that still preserve enough entropy to identify individual users using machine learning classifiers.

The performance of the GCU datasets was in part due to the detail provided by the 'Running Applications' probe values. As I discussed in section 3.2.2, the GCU dataset contained not only foreground and background application names, but also information regarding what types of screen widgets where configured, and application processes that appeared to be specific to the manufacturer of the mobile device.

Validating this approach on a larger number of users with the RICE dataset presented some challenges due to the reduced amount of detail in the 'Running Applications' probe values. In order to compare results from the RICE dataset as directly as possible with the GCU dataset, the 'Running Applications' probe was transformed into a list of open applications that would have appeared in the multi tasking view on iPhone devices. The results when attempting to classify the RICE dataset with all 34 users showed greater than 60% user classification accuracy.

Users were also randomly selected from the RICE dataset to build models with the same user count of 4 and 7 as the GCU datasets. Indeed the models built on a smaller number of users showed higher classification accuracy (compared to all 34 users), resulting in 92% accuracy for 4 users, and 77% for 7 users.

In terms of user privacy, the proposed concatenation and hashing anonymization method in this research is able to increase the input search space for a given hashing algorithm. Effectively this means that even if a malicious user obtained device logs, they would not be able to simply perform a dictionary style attack by hashing popular applications and looking for a match in the logs. Just as a password hash conceals the number of characters, the number of applications is also concealed, requiring the incremental hashing of different combinations of the top N applications. An attacker would have to expend exponentially more time to perform a dictionary attack compared to when data values such as application names where individually hashed, as is often done in public datasets.

Future research would validate this method on larger datasets that contain the same level of fine grained process information as the GCU datasets. Another variation on this method could treat the list of applications or processes as a string of words, and generating topic models, the results of which would be encrypted and provide some type of grouping indicator, possibly in combination with the method outlined in this research. With regards to k-anoniminity, yet another direction would be attempting to use fully homomorphic encryption instead of the encrypting and global recoding technique explored in this research. In a more applied direction, a project could attempt to fully document how a device would offer an API for user logs using a similar type of anonymizing method as outlined in this research.

Bibliography

[1] *iOS 4*. en.wikipedia.org/wiki/IOS_4.

[2] *The LiveLab Project*. http://livelab.recg.rice.edu/.

[3] *R Project for Statistical Computing*. r-project.org/.

[4] *Weka Cross Validation*. weka.wikispaces.com/Generating+cross-validation+folds+(Java+approach).

[5] *Keep your phone safe: How to protect yourself from wireless threats*, 2013. http://consumerreports.org/privacy0613.

[6] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. *NDSS*, February 2015.

[7] Wenliang Du and Zhijun Zhan. Using randomized response techniques for privacy-preserving data mining. *SIGKDD*, August 2013.

[8] Jason Franklin, Damon McCoy, Parisa Tabriz, Vicentiu Neagoe, Jamie Van Randwyk, and Douglas Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. *USENIX Security*, July 2006.

[9] Kayacik H. G. http://www.kayacik.ca/data.html.

[10] Kayacik H. G., Just M., Baillie L., Aspinall D., and Micallef N. Data driven authentication: On the effectiveness of user behaviour modelling with mobile device sensors. *IEEE S&P Symposium*, May 2014.

[11] Craig Gentry. Fully homomorphic encryption using ideal lattices. *STOC*, June 2009.

[12] Craig Gentry and Shai Halevim. Implementing gentry's fully-homomorphic encryption scheme. February 2011.

[13] Fariba Haddadi and A. Nur Zincir-Heywood. Benchmarking the effect of flow exporters and protocol filters on botnet traffic classification. *IEEE Systems Journal*, 10:1390 – 1401, 2016.

[14] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[15] T. Kohno, A. Broido, and k. claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, May 2005.

[16] Alan G. Konheim. *Hashing in Computer Science: Fifty Years of Slicing and Dicing*. Wiley-Interscience, 1 edition, July 2010.

[17] W. D. Maurer. Programming technique: An improved hash code for scatter storage. *Commun. ACM*, 11(1):35–38, January 1968. `http://doi.acm.org/10.1145/362851.362880`.

[18] Robert Morris. Scatter storage techniques. *Commun. ACM*, 11(1):38–44, January 1968. `http://doi.acm.org/10.1145/362851.362882`.

[19] Anthony Quattrone, Tanusri Bhattacharya andLars Kulik, Egemen Tanin, and James Bailey. Is this you? identifying a mobile user using only diagnostic features. *MUM*, November 2014.

[20] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

[21] Mathew J. Schwartz. *Apples iOS Zero-Day PDF Vulnerability Exposed*, July 2011. `https://web.archive.org/web/20110711002808/http://www.informationweek.com/news/231001147`.

[22] Clayton Shepard, Ahmad Rahmati, Chad Tossell, Lin Zhong, and Phillip Kortum. Livelab: measuring wireless networks and smartphone users in the field. *SIGMETRICS Perform. Eval. Rev.*, 38, no. 3, December 2010.

[23] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty*, May 2002.

[24] Brian Voo. *A look Into: The History Of iOS And Its Features*. `hongkiat.com/blog/ios-history/`.

[25] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14:1–37, December 2007.

[26] Jian Xu, Wei Wang, Jian Pei, Xiaoyuan Wang, Baile Shi, and Ada Wai-Chee Fu. Utility-based anonymization using local recoding. *KDD*, August 2006.

[27] Zhiqiang Yang, Sheng Zhong, and Rebecca N. Wright. Privacy-preserving classification of customer data without loss of accuracy. *SIAM International Conference on Data Mining*, April 2005.

Appendix A

GCU v1 Experiments Output

Dataset Features	Min Leafs	MD5 Error	SHA1 Error	SHA256 Error
Time,Apps	3	0.0008%	0.0008%	0.0008%
Time,Apps	5	0.0008%	0.0040%	0.0024%
Time,Apps	7	0.0024%	0.0008%	0.0008%
Time,Apps	10	0.0024%	0.0008%	0.0008%
Time,Apps	15	0.0008%	0.0008%	0.0008%
Time,Apps	22	0.0024%	0.0056%	0.0024%
Time,Apps	32	0.0040%	0.0008%	0.0008%
Time,Apps	46	0.0024%	0.0008%	0.0008%
Time,Apps	68	0.0008%	0.0008%	0.0008%
Time,Apps	100	0.0024%	0.0024%	0.0008%
Time,Cell,Apps	3	0.0000%	0.0016%	0.0000%
Time,Cell,Apps	5	0.0016%	0.0000%	0.0000%
Time,Cell,Apps	7	0.0000%	0.0016%	0.0000%
Time,Cell,Apps	10	0.0016%	0.0016%	0.0016%
Time,Cell,Apps	15	0.0016%	0.0000%	0.0000%
Time,Cell,Apps	22	0.0000%	0.0000%	0.0016%
Time,Cell,Apps	32	0.0016%	0.0000%	0.0000%
Time,Cell,Apps	46	0.0000%	0.0016%	0.0032%
Time,Cell,Apps	68	0.0000%	0.0000%	0.0000%
Time,Cell,Apps	100	0.0016%	0.0032%	0.0000%
Time,Cell,Wifi,Apps	3	0.0000%	0.0000%	0.0016%
Time,Cell,Wifi,Apps	5	0.0000%	0.0000%	0.0000%
Time,Cell,Wifi,Apps	7	0.0016%	0.0000%	0.0016%
Time,Cell,Wifi,Apps	10	0.0000%	0.0032%	0.0000%
Time,Cell,Wifi,Apps	15	0.0000%	0.0000%	0.0000%
Time,Cell,Wifi,Apps	22	0.0016%	0.0000%	0.0016%
Time,Cell,Wifi,Apps	32	0.0000%	0.0000%	0.0000%
Time,Cell,Wifi,Apps	46	0.0000%	0.0000%	0.0016%
Time,Cell,Wifi,Apps	68	0.0000%	0.0016%	0.0000%
Time,Cell,Wifi,Apps	100	0.0000%	0.0000%	0.0000%

Time,Wifi,Apps	3	0.0008%	0.0008%	0.0024%
Time,Wifi,Apps	5	0.0024%	0.0040%	0.0008%
Time,Wifi,Apps	7	0.0008%	0.0008%	0.0008%
Time,Wifi,Apps	10	0.0024%	0.0008%	0.0008%
Time,Wifi,Apps	15	0.0024%	0.0008%	0.0024%
Time,Wifi,Apps	22	0.0024%	0.0008%	0.0008%
Time,Wifi,Apps	32	0.0008%	0.0008%	0.0008%
Time,Wifi,Apps	46	0.0008%	0.0008%	0.0008%
Time,Wifi,Apps	68	0.0008%	0.0008%	0.0008%
Time,Wifi,Apps	100	0.0024%	0.0008%	0.0008%

Table A.1: GCU V1 Experiments Output (Matlab CART)

Table A.2: GCU V1 Baseline Experiments Output (Matlab CART)

Dataset Features	Min Leafs	Error
Time,Apps	3	0.0024%
Time,Apps	5	0.0016%
Time,Apps	7	0.0016%
Time,Apps	10	0.0016%
Time,Apps	15	0.0016%
Time,Apps	22	0.0016%
Time,Apps	32	0.0016%
Time,Apps	46	0.0016%
Time,Apps	68	0.0016%
Time,Apps	100	0.0016%
Time,Cell,Apps	3	0.0805%
Time,Cell,Apps	5	0.0789%
Time,Cell,Apps	7	0.0885%
Time,Cell,Apps	10	0.0789%
Time,Cell,Apps	15	0.0845%
Time,Cell,Apps	22	0.0676%
Time,Cell,Apps	32	0.0708%
Time,Cell,Apps	46	0.0724%
Time,Cell,Apps	68	0.0893%
Time,Cell,Apps	100	0.0773%

Appendix B

GCU v2 Experiments Output

Dataset Features	Min Leafs	MD5 Error	SHA1 Error	SHA256 Error
Time,Apps	3	0.0085%	0.0085%	0.0090%
Time,Apps	5	0.0088%	0.0085%	0.0085%
Time,Apps	7	0.0083%	0.0089%	0.0081%
Time,Apps	10	0.0089%	0.0100%	0.0090%
Time,Apps	15	0.0114%	0.0121%	0.0124%
Time,Apps	22	0.0203%	0.0200%	0.0192%
Time,Apps	32	0.0451%	0.0466%	0.0443%
Time,Apps	46	0.0551%	0.0564%	0.0568%
Time,Apps	68	0.0676%	0.0681%	0.0672%
Time,Apps	100	0.0890%	0.0888%	0.0881%
Time,Cell,Apps	3	0.0000%	0.0000%	0.0000%
Time,Cell,Apps	5	0.0000%	0.0000%	0.0000%
Time,Cell,Apps	7	0.0000%	0.0000%	0.0000%
Time,Cell,Apps	10	0.0000%	0.0000%	0.0000%
Time,Cell,Apps	15	0.0000%	0.0000%	0.0000%
Time,Cell,Apps	22	0.0002%	0.0002%	0.0002%
Time,Cell,Apps	32	0.0010%	0.0011%	0.0013%
Time,Cell,Apps	46	0.0014%	0.0014%	0.0014%
Time,Cell,Apps	68	0.0014%	0.0014%	0.0014%
Time,Cell,Apps	100	0.0014%	0.0014%	0.0014%
Time,Cell,Wifi,Apps	3	0.0000%	0.0000%	0.0000%
Time,Cell,Wifi,Apps	5	0.0000%	0.0000%	0.0000%
Time,Cell,Wifi,Apps	7	0.0000%	0.0000%	0.0000%
Time,Cell,Wifi,Apps	10	0.0000%	0.0000%	0.0000%
Time,Cell,Wifi,Apps	15	0.0000%	0.0000%	0.0001%
Time,Cell,Wifi,Apps	22	0.0003%	0.0002%	0.0002%
Time,Cell,Wifi,Apps	32	0.0009%	0.0010%	0.0009%
Time,Cell,Wifi,Apps	46	0.0016%	0.0016%	0.0016%
Time,Cell,Wifi,Apps	68	0.0021%	0.0021%	0.0020%
Time,Cell,Wifi,Apps	100	0.0031%	0.0031%	0.0031%

Time,Wifi,Apps	3	0.0000%	0.0000%	0.0000%
Time,Wifi,Apps	5	0.0000%	0.0001%	0.0001%
Time,Wifi,Apps	7	0.0001%	0.0001%	0.0000%
Time,Wifi,Apps	10	0.0000%	0.0000%	0.0000%
Time,Wifi,Apps	15	0.0000%	0.0002%	0.0001%
Time,Wifi,Apps	22	0.0004%	0.0006%	0.0003%
Time,Wifi,Apps	32	0.0012%	0.0013%	0.0012%
Time,Wifi,Apps	46	0.0016%	0.0013%	0.0016%
Time,Wifi,Apps	68	0.0014%	0.0016%	0.0014%
Time,Wifi,Apps	100	0.0014%	0.0012%	0.0014%

Table B.1: GCU V2 Experiments Output (Matlab CART)

Table B.2: GCU V2 Baseline Experiments Output (Matlab CART)

Dataset Features	Min Leafs	Error
Time,Apps	3	0.0011%
Time,Apps	5	0.0012%
Time,Apps	7	0.0011%
Time,Apps	10	0.0011%
Time,Apps	15	0.0014%
Time,Apps	22	0.0016%
Time,Apps	32	0.0015%
Time,Apps	46	0.0014%
Time,Apps	68	0.0014%
Time,Apps	100	0.0014%
Time,Cell,Apps	3	0.0019%
Time,Cell,Apps	5	0.0019%
Time,Cell,Apps	7	0.0017%
Time,Cell,Apps	10	0.0018%
Time,Cell,Apps	15	0.0017%
Time,Cell,Apps	22	0.0019%
Time,Cell,Apps	32	0.0024%
Time,Cell,Apps	46	0.0026%
Time,Cell,Apps	68	0.0026%
Time,Cell,Apps	100	0.0032%

Appendix C

RICE Experiments Output

Dataset Features	Users	MD5 Accuracy	SHA1 Accuracy	SHA256 Accuracy
day,time,apps	4	49.8698 %	49.8698 %	49.8698 %
day,time,apps	4	53.5298 %	53.5298 %	53.5298 %
day,time,apps	4	55.1926 %	55.1926 %	55.1926 %
day,time,apps	7	42.8203 %	42.8203 %	42.8203 %
day,time,apps	7	38.9298 %	38.9298 %	38.9298 %
day,time,apps	7	42.0554 %	42.0554 %	42.0554 %
day,time,apps	14	26.3153 %	26.3153 %	26.3153 %
day,time,apps	14	26.2357 %	26.2357 %	26.2357 %
day,time,apps	14	25.838 %	25.838 %	25.838 %
day,time,apps	25	19.0279 %	19.0279 %	19.0279 %
day,time,apps	25	19.2794 %	19.2794 %	19.2794 %
day,time,apps	25	20.2535 %	20.2535 %	20.2535 %
day,time,apps	34	15.7804 %	15.7804 %	15.7804 %
day,time,cell,apps	4	93.4539 %	93.4539 %	93.4539 %
day,time,cell,apps	4	96.0432 %	96.0432 %	96.0432 %
day,time,cell,apps	4	92.1222 %	92.1222 %	92.1222 %
day,time,cell,apps	7	93.8342 %	93.8342 %	93.8342 %
day,time,cell,apps	7	78.2305 %	78.2305 %	78.2305 %
day,time,cell,apps	7	82.0762 %	82.0762 %	82.0762 %
day,time,cell,apps	14	76.4301 %	76.4301 %	76.4301 %
day,time,cell,apps	14	76.3739 %	76.3739 %	76.3739 %
day,time,cell,apps	14	80.1445 %	80.1445 %	80.1445 %
day,time,cell,apps	25	69.9857 %	69.9857 %	69.9857 %
day,time,cell,apps	25	69.9028 %	69.9028 %	69.9028 %
day,time,cell,apps	25	70.3911 %	70.3911 %	70.3911 %
day,time,cell,apps	34	65.789 %	65.789 %	65.789 %
day,time,cell,wifi,apps	4	95.2834 %	95.2834 %	95.2834 %
day,time,cell,wifi,apps	4	96.0432 %	96.0432 %	96.0432 %
day,time,cell,wifi,apps	4	92.9454 %	92.9454 %	92.9454 %
day,time,cell,wifi,apps	7	94.1883 %	94.1883 %	94.1883 %
day,time,cell,wifi,apps	7	80.4689 %	80.4689 %	80.4689 %
day,time,cell,wifi,apps	7	82.0762 %	82.0762 %	82.0762 %
day,time,cell,wifi,apps	14	77.3864 %	77.3864 %	77.3864 %
day,time,cell,wifi,apps	14	77.5385 %	77.5385 %	77.5385 %

day,time,cell,wifi,apps	14	81.3085 %	81.3085 %	81.3085 %
day,time,cell,wifi,apps	25	72.3149 %	72.3149 %	72.3149 %
day,time,cell,wifi,apps	25	72.2411 %	72.2411 %	72.2411 %
day,time,cell,wifi,apps	25	72.551 %	72.551 %	72.551 %
day,time,cell,wifi,apps	34	68.2777 %	68.2777 %	68.2777 %
day,time,wifi,apps	4	54.9031 %	54.9031 %	54.9031 %
day,time,wifi,apps	4	53.5298 %	53.5298 %	53.5298 %
day,time,wifi,apps	4	57.5394 %	57.5394 %	57.5394 %
day,time,wifi,apps	7	46.2414 %	46.2414 %	46.2414 %
day,time,wifi,apps	7	43.1957 %	43.1957 %	43.1957 %
day,time,wifi,apps	7	42.0554 %	42.0554 %	42.0554 %
day,time,wifi,apps	14	27.9534 %	27.9534 %	27.9534 %
day,time,wifi,apps	14	28.7838 %	28.7838 %	28.7838 %
day,time,wifi,apps	14	27.7146 %	27.7146 %	27.7146 %
day,time,wifi,apps	25	23.1844 %	23.1844 %	23.1844 %
day,time,wifi,apps	25	23.5657 %	23.5657 %	23.5657 %
day,time,wifi,apps	25	24.3977 %	24.3977 %	24.3977 %
day,time,wifi,apps	34	19.9632 %	19.9632 %	19.9632 %
time,apps	4	49.0163 %	49.0163 %	49.0163 %
time,apps	4	50.9146 %	50.9146 %	50.9146 %
time,apps	4	52.5209 %	52.5209 %	52.5209 %
time,apps	7	34.8724 %	34.8724 %	34.8724 %
time,apps	7	37.2801 %	37.2801 %	37.2801 %
time,apps	7	38.4493 %	38.4493 %	38.4493 %
time,apps	14	25.7782 %	25.7782 %	25.7782 %
time,apps	14	23.2494 %	23.2494 %	23.2494 %
time,apps	14	23.2482 %	23.2482 %	23.2482 %
time,apps	25	16.7321 %	16.7321 %	16.7321 %
time,apps	25	16.75 %	16.75 %	16.75 %
time,apps	25	17.663 %	17.663 %	17.663 %
time,apps	34	13.279 %	13.279 %	13.279 %
time,cell,apps	4	88.2146 %	88.2146 %	88.2146 %
time,cell,apps	4	95.2407 %	95.2407 %	95.2407 %
time,cell,apps	4	90.8015 %	90.8015 %	90.8015 %
time,cell,apps	7	74.3889 %	74.3889 %	74.3889 %
time,cell,apps	7	76.2498 %	76.2498 %	76.2498 %
time,cell,apps	7	78.9663 %	78.9663 %	78.9663 %
time,cell,apps	14	70.9949 %	70.9949 %	70.9949 %
time,cell,apps	14	73.2004 %	73.2004 %	73.2004 %
time,cell,apps	14	77.5147 %	77.5147 %	77.5147 %
time,cell,apps	25	64.861 %	64.861 %	64.861 %

time,cell,apps	25	66.2782 %	66.2782 %	66.2782 %
time,cell,apps	25	66.4869 %	66.4869 %	66.4869 %
time,cell,apps	34	61.9527 %	61.9527 %	61.9527 %
time,cell,wifi,apps	4	89.3201 %	89.3201 %	89.3201 %
time,cell,wifi,apps	4	95.2407 %	95.2407 %	95.2407 %
time,cell,wifi,apps	4	91.6323 %	91.6323 %	91.6323 %
time,cell,wifi,apps	7	74.3889 %	74.3889 %	74.3889 %
time,cell,wifi,apps	7	78.712 %	78.712 %	78.712 %
time,cell,wifi,apps	7	78.9663 %	78.9663 %	78.9663 %
time,cell,wifi,apps	14	73.0573 %	73.0573 %	73.0573 %
time,cell,wifi,apps	14	74.3575 %	74.3575 %	74.3575 %
time,cell,wifi,apps	14	78.8166 %	78.8166 %	78.8166 %
time,cell,wifi,apps	25	68.0834 %	68.0834 %	68.0834 %
time,cell,wifi,apps	25	68.696 %	68.696 %	68.696 %
time,cell,wifi,apps	25	68.7856 %	68.7856 %	68.7856 %
time,cell,wifi,apps	34	64.5358 %	64.5358 %	64.5358 %
time,wifi,apps	4	51.2434 %	51.2434 %	51.2434 %
time,wifi,apps	4	50.9146 %	50.9146 %	50.9146 %
time,wifi,apps	4	55.0628 %	55.0628 %	55.0628 %
time,wifi,apps	7	34.8724 %	34.8724 %	34.8724 %
time,wifi,apps	7	41.6965 %	41.6965 %	41.6965 %
time,wifi,apps	7	38.4493 %	38.4493 %	38.4493 %
time,wifi,apps	14	28.472 %	28.472 %	28.472 %
time,wifi,apps	14	25.8803 %	25.8803 %	25.8803 %
time,wifi,apps	14	25.4114 %	25.4114 %	25.4114 %
time,wifi,apps	25	21.3676 %	21.3676 %	21.3676 %
time,wifi,apps	25	21.2661 %	21.2661 %	21.2661 %
time,wifi,apps	25	21.9965 %	21.9965 %	21.9965 %
time,wifi,apps	34	17.9312 %	17.9312 %	17.9312 %

Table C.1: RICE c4.5 Experiment Output (WEKA C4.5)

Hashing Algorithm	Dataset Features	Users	Data Instances	Leaves	Tree Size	Accuracy
md5	day,time,apps	4	150915	1230	2459	49.8698 %
md5	day,time,apps	4	145523	1109	2217	55.1926 %
md5	day,time,apps	4	133718	1088	2175	53.5298 %
md5	day,time,apps	7	245270	2149	4297	38.9298 %
md5	day,time,apps	7	233546	2305	4609	42.8203 %

md5	day,time,apps	7	132801	1177	2353	42.0554 %
md5	day,time,apps	14	430033	7050	14099	26.2357 %
md5	day,time,apps	14	419843	5998	11995	25.838 %
md5	day,time,apps	14	404023	5537	11073	26.3153 %
md5	day,time,apps	25	798612	16945	33889	20.2535 %
md5	day,time,apps	25	790611	17313	34625	19.2794 %
md5	day,time,apps	25	790985	17451	34901	19.0279 %
md5	day,time,apps	34	1126520	25555	51109	15.7804 %
md5	day,time,cell,apps	4	150915	328	655	93.4539 %
md5	day,time,cell,apps	4	145523	501	1001	92.1222 %
md5	day,time,cell,apps	4	133718	347	693	96.0432 %
md5	day,time,cell,apps	7	245270	1493	2985	78.2305 %
md5	day,time,cell,apps	7	233546	752	1503	93.8342 %
md5	day,time,cell,apps	7	132801	1010	2019	82.0762 %
md5	day,time,cell,apps	14	430033	3769	7537	76.3739 %
md5	day,time,cell,apps	14	419843	3230	6459	80.1445 %
md5	day,time,cell,apps	14	404023	3261	6521	76.4301 %
md5	day,time,cell,apps	25	798612	8614	17227	70.3911 %
md5	day,time,cell,apps	25	790611	8166	16331	69.9028 %
md5	day,time,cell,apps	25	790985	7882	15763	69.9857 %
md5	day,time,cell,apps	34	1126520	12867	25733	65.789 %
md5	day,time,cell,wifi,apps	4	150915	265	529	95.2834 %
md5	day,time,cell,wifi,apps	4	145523	463	925	92.9454 %
md5	day,time,cell,wifi,apps	4	133718	347	693	96.0432 %
md5	day,time,cell,wifi,apps	7	245270	1542	3083	80.4689 %
md5	day,time,cell,wifi,apps	7	233546	754	1507	94.1883 %
md5	day,time,cell,wifi,apps	7	132801	1010	2019	82.0762 %
md5	day,time,cell,wifi,apps	14	430033	3647	7293	77.5385 %
md5	day,time,cell,wifi,apps	14	419843	3032	6063	81.3085 %
md5	day,time,cell,wifi,apps	14	404023	3224	6447	77.3864 %
md5	day,time,cell,wifi,apps	25	798612	8426	16851	72.551 %
md5	day,time,cell,wifi,apps	25	790611	7828	15655	72.2411 %
md5	day,time,cell,wifi,apps	25	790985	7804	15607	72.3149 %
md5	day,time,cell,wifi,apps	34	1126520	12343	24685	68.2777 %
md5	day,time,wifi,apps	4	150915	1225	2449	54.9031 %
md5	day,time,wifi,apps	4	145523	1052	2103	57.5394 %
md5	day,time,wifi,apps	4	133718	1088	2175	53.5298 %
md5	day,time,wifi,apps	7	245270	2168	4335	43.1957 %
md5	day,time,wifi,apps	7	233546	2244	4487	46.2414 %
md5	day,time,wifi,apps	7	132801	1177	2353	42.0554 %
md5	day,time,wifi,apps	14	430033	6420	12839	28.7838 %

md5	day,time,wifi,apps	14	419843	5817	11633	27.7146 %
md5	day,time,wifi,apps	14	404023	5190	10379	27.9534 %
md5	day,time,wifi,apps	25	798612	14496	28991	24.3977 %
md5	day,time,wifi,apps	25	790611	15138	30275	23.5657 %
md5	day,time,wifi,apps	25	790985	15907	31813	23.1844 %
md5	day,time,wifi,apps	34	1126520	23185	46369	19.9632 %
md5	time,apps	4	145523	181	361	52.5209 %
md5	time,apps	4	148981	168	335	49.0163 %
md5	time,apps	4	133718	173	345	50.9146 %
md5	time,apps	7	245270	473	945	37.2801 %
md5	time,apps	7	250946	739	1477	34.8724 %
md5	time,apps	7	132801	200	399	38.4493 %
md5	time,apps	14	443626	3889	7777	25.7782 %
md5	time,apps	14	430033	5054	10107	23.2494 %
md5	time,apps	14	419843	4586	9171	23.2482 %
md5	time,apps	25	798612	12609	25217	17.663 %
md5	time,apps	25	790611	12928	25855	16.75 %
md5	time,apps	25	772974	12784	25567	16.7321 %
md5	time,apps	34	1126520	19572	39143	13.279 %
md5	time,cell,apps	4	145523	244	487	90.8015 %
md5	time,cell,apps	4	148981	219	437	88.2146 %
md5	time,cell,apps	4	133718	282	563	95.2407 %
md5	time,cell,apps	7	245270	654	1307	76.2498 %
md5	time,cell,apps	7	250946	967	1933	74.3889 %
md5	time,cell,apps	7	132801	558	1115	78.9663 %
md5	time,cell,apps	14	443626	2234	4467	70.9949 %
md5	time,cell,apps	14	430033	2146	4291	73.2004 %
md5	time,cell,apps	14	419843	1962	3923	77.5147 %
md5	time,cell,apps	25	798612	5558	11115	66.4869 %
md5	time,cell,apps	25	790611	5630	11259	66.2782 %
md5	time,cell,apps	25	772974	5917	11833	64.861 %
md5	time,cell,apps	34	1126520	9826	19651	61.9527 %
md5	time,cell,wifi,apps	4	145523	244	487	91.6323 %
md5	time,cell,wifi,apps	4	148981	211	421	89.3201 %
md5	time,cell,wifi,apps	4	133718	282	563	95.2407 %
md5	time,cell,wifi,apps	7	245270	681	1361	78.712 %
md5	time,cell,wifi,apps	7	250946	967	1933	74.3889 %
md5	time,cell,wifi,apps	7	132801	558	1115	78.9663 %
md5	time,cell,wifi,apps	14	443626	2142	4283	73.0573 %
md5	time,cell,wifi,apps	14	430033	2160	4319	74.3575 %
md5	time,cell,wifi,apps	14	419843	1944	3887	78.8166 %

md5	time,cell,wifi,apps	25	798612	5437	10873	68.7856 %
md5	time,cell,wifi,apps	25	790611	5237	10473	68.696 %
md5	time,cell,wifi,apps	25	772974	5280	10559	68.0834 %
md5	time,cell,wifi,apps	34	1126520	9240	18479	64.5358 %
md5	time,wifi,apps	4	145523	176	351	55.0628 %
md5	time,wifi,apps	4	148981	167	333	51.2434 %
md5	time,wifi,apps	4	133718	173	345	50.9146 %
md5	time,wifi,apps	7	245270	494	987	41.6965 %
md5	time,wifi,apps	7	250946	739	1477	34.8724 %
md5	time,wifi,apps	7	132801	200	399	38.4493 %
md5	time,wifi,apps	14	443626	3496	6991	28.472 %
md5	time,wifi,apps	14	430033	4517	9033	25.8803 %
md5	time,wifi,apps	14	419843	4006	8011	25.4114 %
md5	time,wifi,apps	25	798612	11970	23939	21.9965 %
md5	time,wifi,apps	25	790611	12320	24639	21.2661 %
md5	time,wifi,apps	25	772974	12232	24463	21.3676 %
md5	time,wifi,apps	34	1126520	18997	37993	17.9312 %
sha1	day,time,apps	4	150915	1230	2459	49.8698 %
sha1	day,time,apps	4	145523	1109	2217	55.1926 %
sha1	day,time,apps	4	133718	1088	2175	53.5298 %
sha1	day,time,apps	7	245270	2149	4297	38.9298 %
sha1	day,time,apps	7	233546	2305	4609	42.8203 %
sha1	day,time,apps	7	132801	1177	2353	42.0554 %
sha1	day,time,apps	14	430033	7050	14099	26.2357 %
sha1	day,time,apps	14	419843	5998	11995	25.838 %
sha1	day,time,apps	14	404023	5537	11073	26.3153 %
sha1	day,time,apps	25	798612	16945	33889	20.2535 %
sha1	day,time,apps	25	790611	17313	34625	19.2794 %
sha1	day,time,apps	25	790985	17451	34901	19.0279 %
sha1	day,time,apps	34	1126520	25555	51109	15.7804 %
sha1	day,time,cell,apps	4	150915	328	655	93.4539 %
sha1	day,time,cell,apps	4	145523	501	1001	92.1222 %
sha1	day,time,cell,apps	4	133718	347	693	96.0432 %
sha1	day,time,cell,apps	7	245270	1493	2985	78.2305 %
sha1	day,time,cell,apps	7	233546	752	1503	93.8342 %
sha1	day,time,cell,apps	7	132801	1010	2019	82.0762 %
sha1	day,time,cell,apps	14	430033	3769	7537	76.3739 %
sha1	day,time,cell,apps	14	419843	3230	6459	80.1445 %
sha1	day,time,cell,apps	14	404023	3261	6521	76.4301 %
sha1	day,time,cell,apps	25	798612	8614	17227	70.3911 %
sha1	day,time,cell,apps	25	790611	8166	16331	69.9028 %

sha1	day,time,cell,apps	25	790985	7882	15763	69.9857 %
sha1	day,time,cell,apps	34	1126520	12867	25733	65.789 %
sha1	day,time,cell,wifi,apps	4	150915	265	529	95.2834 %
sha1	day,time,cell,wifi,apps	4	145523	463	925	92.9454 %
sha1	day,time,cell,wifi,apps	4	133718	347	693	96.0432 %
sha1	day,time,cell,wifi,apps	7	245270	1542	3083	80.4689 %
sha1	day,time,cell,wifi,apps	7	233546	754	1507	94.1883 %
sha1	day,time,cell,wifi,apps	7	132801	1010	2019	82.0762 %
sha1	day,time,cell,wifi,apps	14	430033	3647	7293	77.5385 %
sha1	day,time,cell,wifi,apps	14	419843	3032	6063	81.3085 %
sha1	day,time,cell,wifi,apps	14	404023	3224	6447	77.3864 %
sha1	day,time,cell,wifi,apps	25	798612	8426	16851	72.551 %
sha1	day,time,cell,wifi,apps	25	790611	7828	15655	72.2411 %
sha1	day,time,cell,wifi,apps	25	790985	7804	15607	72.3149 %
sha1	day,time,cell,wifi,apps	34	1126520	12343	24685	68.2777 %
sha1	day,time,wifi,apps	4	150915	1225	2449	54.9031 %
sha1	day,time,wifi,apps	4	145523	1052	2103	57.5394 %
sha1	day,time,wifi,apps	4	133718	1088	2175	53.5298 %
sha1	day,time,wifi,apps	7	245270	2168	4335	43.1957 %
sha1	day,time,wifi,apps	7	233546	2244	4487	46.2414 %
sha1	day,time,wifi,apps	7	132801	1177	2353	42.0554 %
sha1	day,time,wifi,apps	14	430033	6420	12839	28.7838 %
sha1	day,time,wifi,apps	14	419843	5817	11633	27.7146 %
sha1	day,time,wifi,apps	14	404023	5190	10379	27.9534 %
sha1	day,time,wifi,apps	25	798612	14496	28991	24.3977 %
sha1	day,time,wifi,apps	25	790611	15138	30275	23.5657 %
sha1	day,time,wifi,apps	25	790985	15907	31813	23.1844 %
sha1	day,time,wifi,apps	34	1126520	23185	46369	19.9632 %
sha1	time,apps	4	145523	181	361	52.5209 %
sha1	time,apps	4	148981	168	335	49.0163 %
sha1	time,apps	4	133718	173	345	50.9146 %
sha1	time,apps	7	245270	473	945	37.2801 %
sha1	time,apps	7	250946	739	1477	34.8724 %
sha1	time,apps	7	132801	200	399	38.4493 %
sha1	time,apps	14	443626	3889	7777	25.7782 %
sha1	time,apps	14	430033	5054	10107	23.2494 %
sha1	time,apps	14	419843	4586	9171	23.2482 %
sha1	time,apps	25	798612	12609	25217	17.663 %
sha1	time,apps	25	790611	12928	25855	16.75 %
sha1	time,apps	25	772974	12784	25567	16.7321 %
sha1	time,apps	34	1126520	19572	39143	13.279 %

sha1	time,cell,apps	4	145523	244	487	90.8015 %
sha1	time,cell,apps	4	148981	219	437	88.2146 %
sha1	time,cell,apps	4	133718	282	563	95.2407 %
sha1	time,cell,apps	7	245270	654	1307	76.2498 %
sha1	time,cell,apps	7	250946	967	1933	74.3889 %
sha1	time,cell,apps	7	132801	558	1115	78.9663 %
sha1	time,cell,apps	14	443626	2234	4467	70.9949 %
sha1	time,cell,apps	14	430033	2146	4291	73.2004 %
sha1	time,cell,apps	14	419843	1962	3923	77.5147 %
sha1	time,cell,apps	25	798612	5558	11115	66.4869 %
sha1	time,cell,apps	25	790611	5630	11259	66.2782 %
sha1	time,cell,apps	25	772974	5917	11833	64.861 %
sha1	time,cell,apps	34	1126520	9826	19651	61.9527 %
sha1	time,cell,wifi,apps	4	145523	244	487	91.6323 %
sha1	time,cell,wifi,apps	4	148981	211	421	89.3201 %
sha1	time,cell,wifi,apps	4	133718	282	563	95.2407 %
sha1	time,cell,wifi,apps	7	245270	681	1361	78.712 %
sha1	time,cell,wifi,apps	7	250946	967	1933	74.3889 %
sha1	time,cell,wifi,apps	7	132801	558	1115	78.9663 %
sha1	time,cell,wifi,apps	14	443626	2142	4283	73.0573 %
sha1	time,cell,wifi,apps	14	430033	2160	4319	74.3575 %
sha1	time,cell,wifi,apps	14	419843	1944	3887	78.8166 %
sha1	time,cell,wifi,apps	25	798612	5437	10873	68.7856 %
sha1	time,cell,wifi,apps	25	790611	5237	10473	68.696 %
sha1	time,cell,wifi,apps	25	772974	5280	10559	68.0834 %
sha1	time,cell,wifi,apps	34	1126520	9240	18479	64.5358 %
sha1	time,wifi,apps	4	145523	176	351	55.0628 %
sha1	time,wifi,apps	4	148981	167	333	51.2434 %
sha1	time,wifi,apps	4	133718	173	345	50.9146 %
sha1	time,wifi,apps	7	245270	494	987	41.6965 %
sha1	time,wifi,apps	7	250946	739	1477	34.8724 %
sha1	time,wifi,apps	7	132801	200	399	38.4493 %
sha1	time,wifi,apps	14	443626	3496	6991	28.472 %
sha1	time,wifi,apps	14	430033	4517	9033	25.8803 %
sha1	time,wifi,apps	14	419843	4006	8011	25.4114 %
sha1	time,wifi,apps	25	798612	11970	23939	21.9965 %
sha1	time,wifi,apps	25	790611	12320	24639	21.2661 %
sha1	time,wifi,apps	25	772974	12232	24463	21.3676 %
sha1	time,wifi,apps	34	1126520	18997	37993	17.9312 %
sha256	day,time,apps	4	150915	1230	2459	49.8698 %
sha256	day,time,apps	4	145523	1109	2217	55.1926 %

sha256	day,time,apps	4	133718	1088	2175	53.5298 %
sha256	day,time,apps	7	245270	2149	4297	38.9298 %
sha256	day,time,apps	7	233546	2305	4609	42.8203 %
sha256	day,time,apps	7	132801	1177	2353	42.0554 %
sha256	day,time,apps	14	430033	7050	14099	26.2357 %
sha256	day,time,apps	14	419843	5998	11995	25.838 %
sha256	day,time,apps	14	404023	5537	11073	26.3153 %
sha256	day,time,apps	25	798612	16945	33889	20.2535 %
sha256	day,time,apps	25	790611	17313	34625	19.2794 %
sha256	day,time,apps	25	790985	17451	34901	19.0279 %
sha256	day,time,apps	34	1126520	25555	51109	15.7804 %
sha256	day,time,cell,apps	4	150915	328	655	93.4539 %
sha256	day,time,cell,apps	4	145523	501	1001	92.1222 %
sha256	day,time,cell,apps	4	133718	347	693	96.0432 %
sha256	day,time,cell,apps	7	245270	1493	2985	78.2305 %
sha256	day,time,cell,apps	7	233546	752	1503	93.8342 %
sha256	day,time,cell,apps	7	132801	1010	2019	82.0762 %
sha256	day,time,cell,apps	14	430033	3769	7537	76.3739 %
sha256	day,time,cell,apps	14	419843	3230	6459	80.1445 %
sha256	day,time,cell,apps	14	404023	3261	6521	76.4301 %
sha256	day,time,cell,apps	25	798612	8614	17227	70.3911 %
sha256	day,time,cell,apps	25	790611	8166	16331	69.9028 %
sha256	day,time,cell,apps	25	790985	7882	15763	69.9857 %
sha256	day,time,cell,apps	34	1126520	12867	25733	65.789 %
sha256	day,time,cell,wifi,apps	4	150915	265	529	95.2834 %
sha256	day,time,cell,wifi,apps	4	145523	463	925	92.9454 %
sha256	day,time,cell,wifi,apps	4	133718	347	693	96.0432 %
sha256	day,time,cell,wifi,apps	7	245270	1542	3083	80.4689 %
sha256	day,time,cell,wifi,apps	7	233546	754	1507	94.1883 %
sha256	day,time,cell,wifi,apps	7	132801	1010	2019	82.0762 %
sha256	day,time,cell,wifi,apps	14	430033	3647	7293	77.5385 %
sha256	day,time,cell,wifi,apps	14	419843	3032	6063	81.3085 %
sha256	day,time,cell,wifi,apps	14	404023	3224	6447	77.3864 %
sha256	day,time,cell,wifi,apps	25	798612	8426	16851	72.551 %
sha256	day,time,cell,wifi,apps	25	790611	7828	15655	72.2411 %
sha256	day,time,cell,wifi,apps	25	790985	7804	15607	72.3149 %
sha256	day,time,cell,wifi,apps	34	1126520	12343	24685	68.2777 %
sha256	day,time,wifi,apps	4	150915	1225	2449	54.9031 %
sha256	day,time,wifi,apps	4	145523	1052	2103	57.5394 %
sha256	day,time,wifi,apps	4	133718	1088	2175	53.5298 %
sha256	day,time,wifi,apps	7	245270	2168	4335	43.1957 %

sha256	day,time,wifi,apps	7	233546	2244	4487	46.2414 %
sha256	day,time,wifi,apps	7	132801	1177	2353	42.0554 %
sha256	day,time,wifi,apps	14	430033	6420	12839	28.7838 %
sha256	day,time,wifi,apps	14	419843	5817	11633	27.7146 %
sha256	day,time,wifi,apps	14	404023	5190	10379	27.9534 %
sha256	day,time,wifi,apps	25	798612	14496	28991	24.3977 %
sha256	day,time,wifi,apps	25	790611	15138	30275	23.5657 %
sha256	day,time,wifi,apps	25	790985	15907	31813	23.1844 %
sha256	day,time,wifi,apps	34	1126520	23185	46369	19.9632 %
sha256	time,apps	4	145523	181	361	52.5209 %
sha256	time,apps	4	148981	168	335	49.0163 %
sha256	time,apps	4	133718	173	345	50.9146 %
sha256	time,apps	7	245270	473	945	37.2801 %
sha256	time,apps	7	250946	739	1477	34.8724 %
sha256	time,apps	7	132801	200	399	38.4493 %
sha256	time,apps	14	443626	3889	7777	25.7782 %
sha256	time,apps	14	430033	5054	10107	23.2494 %
sha256	time,apps	14	419843	4586	9171	23.2482 %
sha256	time,apps	25	798612	12609	25217	17.663 %
sha256	time,apps	25	790611	12928	25855	16.75 %
sha256	time,apps	25	772974	12784	25567	16.7321 %
sha256	time,apps	34	1126520	19572	39143	13.279 %
sha256	time,cell,apps	4	145523	244	487	90.8015 %
sha256	time,cell,apps	4	148981	219	437	88.2146 %
sha256	time,cell,apps	4	133718	282	563	95.2407 %
sha256	time,cell,apps	7	245270	654	1307	76.2498 %
sha256	time,cell,apps	7	250946	967	1933	74.3889 %
sha256	time,cell,apps	7	132801	558	1115	78.9663 %
sha256	time,cell,apps	14	443626	2234	4467	70.9949 %
sha256	time,cell,apps	14	430033	2146	4291	73.2004 %
sha256	time,cell,apps	14	419843	1962	3923	77.5147 %
sha256	time,cell,apps	25	798612	5558	11115	66.4869 %
sha256	time,cell,apps	25	790611	5630	11259	66.2782 %
sha256	time,cell,apps	25	772974	5917	11833	64.861 %
sha256	time,cell,apps	34	1126520	9826	19651	61.9527 %
sha256	time,cell,wifi,apps	4	145523	244	487	91.6323 %
sha256	time,cell,wifi,apps	4	148981	211	421	89.3201 %
sha256	time,cell,wifi,apps	4	133718	282	563	95.2407 %
sha256	time,cell,wifi,apps	7	245270	681	1361	78.712 %
sha256	time,cell,wifi,apps	7	250946	967	1933	74.3889 %
sha256	time,cell,wifi,apps	7	132801	558	1115	78.9663 %

sha256	time,cell,wifi,apps	14	443626	2142	4283	73.0573 %
sha256	time,cell,wifi,apps	14	430033	2160	4319	74.3575 %
sha256	time,cell,wifi,apps	14	419843	1944	3887	78.8166 %
sha256	time,cell,wifi,apps	25	798612	5437	10873	68.7856 %
sha256	time,cell,wifi,apps	25	790611	5237	10473	68.696 %
sha256	time,cell,wifi,apps	25	772974	5280	10559	68.0834 %
sha256	time,cell,wifi,apps	34	1126520	9240	18479	64.5358 %
sha256	time,wifi,apps	4	145523	176	351	55.0628 %
sha256	time,wifi,apps	4	148981	167	333	51.2434 %
sha256	time,wifi,apps	4	133718	173	345	50.9146 %
sha256	time,wifi,apps	7	245270	494	987	41.6965 %
sha256	time,wifi,apps	7	250946	739	1477	34.8724 %
sha256	time,wifi,apps	7	132801	200	399	38.4493 %
sha256	time,wifi,apps	14	443626	3496	6991	28.472 %
sha256	time,wifi,apps	14	430033	4517	9033	25.8803 %
sha256	time,wifi,apps	14	419843	4006	8011	25.4114 %
sha256	time,wifi,apps	25	798612	11970	23939	21.9965 %
sha256	time,wifi,apps	25	790611	12320	24639	21.2661 %
sha256	time,wifi,apps	25	772974	12232	24463	21.3676 %
sha256	time,wifi,apps	34	1126520	18997	37993	17.9312 %

Table C.4: RICE Tree Size vs. Number of Users (WEKA C4.5)

Table C.2: RICE c5.0 Decision Tree Output (WEKA C4.5)

Model Build Time (seconds)	Training Accuracy	Test Accuracy	Dataset
407.38	16.39%	16.42%	sha256_time+wifi+apps_34_users
385.16	65.40%	65.51%	sha256_time+cell+wifi+apps_34_users
358.58	62.93%	62.92%	sha256_time+cell+apps_34_users
375.96	11.23%	11.27%	sha256_time+apps_34_users
502.47	19.96%	20.12%	sha256_day+time+wifi+apps_34_users
426.86	72.69%	**73.03%**	sha256_day+time+cell+wifi+apps_34_users
394.16	70.63%	71.08%	sha256_day+time+cell+apps_34_users
514.12	17.37%	17.99%	sha256_day+time+apps_34_users
445.69	16.40%	16.42%	sha1_time+wifi+apps_34_users
426.23	65.36%	65.64%	sha1_time+cell+wifi+apps_34_users
393.90	62.82%	63.20%	sha1_time+cell+apps_34_users
424.30	11.25%	11.27%	sha1_time+apps_34_users
544.24	19.92%	20.16%	sha1_day+time+wifi+apps_34_users
464.48	72.66%	**73.04%**	sha1_day+time+cell+wifi+apps_34_users
399.56	70.64%	70.99%	sha1_day+time+cell+apps_34_users
505.71	17.37%	17.85%	sha1_day+time+apps_34_users
441.23	16.46%	16.36%	md5_time+wifi+apps_34_users
412.50	65.44%	65.63%	md5_time+cell+wifi+apps_34_users
383.52	62.89%	63.05%	md5_time+cell+apps_34_users
407.10	11.26%	11.29%	md5_time+apps_34_users
538.10	19.93%	20.12%	md5_day+time+wifi+apps_34_users
452.79	72.73%	**72.81%**	md5_day+time+cell+wifi+apps_34_users
426.87	70.67%	71.07%	md5_day+time+cell+apps_34_users
523.90	17.36%	17.96%	md5_day+time+apps_34_users
Model with 'Year' and 'Month' features included			
269.65	81.73%	**82.08%**	sha256_y+m+day+time+wifi+apps_34_users

Table C.3: RICE Baseline Experiments Output (Matlab CART)

Dataset Features	Min Leafs	Error
Time,Apps	22	1.4035%
Time,Apps	32	1.3766%
Time,Apps	46	1.3543%
Time,Apps	68	1.3490%
Time,Apps	100	1.3499%
Time,Apps	100	1.3493%
Time,Apps	158	1.3508%
Time,Apps	251	1.3525%
Time,Apps	398	1.3531%
Time,Apps	631	1.3531%
Time,Apps	1000	1.3531%
Time,Cell,Apps	100	0.7167%
Time,Cell,Apps	158	0.7624%
Time,Cell,Apps	251	0.8453%
Time,Cell,Apps	398	0.8944%
Time,Cell,Apps	631	1.0244%
Time,Cell,Apps	1000	1.2170%